All_D_B_Integrazione_RIS-PACS_sistemi_informativi
Specifiche per l'integrazione con le infrastrutture e il sistema informativo
di AUSL della Romagna

Cap. Σ.1. – Premessa

Per ogni sistema di integrazione descritto, il fornitore dovrà produrre adeguata e completa documentazione su come intenda assolvere alle indicazioni ivi riportate, secondo le peculiarità del sistema proposto. Si precisa che l'Azienda è in fase di riprogettazione di tutto il proprio sistema infrastrutturale di base, pertanto in alcuni casi si rinvia, per la fornitura di precise specifiche, alla fase di progetto esecutivo.

Cap. Σ.2. – Gestione anagrafica

Il Sistema proposto non deve costituire una nuova anagrafe centralizzata autonoma, ma deve porsi in totale dipendenza (slave) dal sistema di anagrafe EMPI dell'Azienda relativamente a tutte le tipologie di dato presenti. L'anagrafe aziendale è a sua volta slave del sistema EMPI regionale.

L'Azienda USL dispone di un sistema di Enterprise Master Patient Index (EMPI), che, nei progetti evolutivi in ambito regionale, sarà asservito al corrispondente EMPI regionale che costituirà master per tulle le aziende regionali. Pur tenendo conto di questa ulteriore specificità, il sistema aziendale consente:

- l'esecuzione di operazioni di fusione di posizioni anagrafiche relative a uno stesso paziente associando alla posizione corretta un identificativo master e collegando a questo gli identificativi delle posizioni fuse (merge);
- la notifica a tutti gli applicativi integrati nel sistema delle variazioni anagrafiche.
- l'identificazione univoca del paziente per mezzo di un identificativo univoco (PK) che ne costituisce la chiave primaria: da ciò discende la possibilità di aggregare correttamente i referti e i risultati all'interno dei sistemi aziendali in cooperazione applicativa (es. nel data-repository clinico).
- la definizione e il mantenimento di anagrafiche comuni che consente l'integrazione di diversi applicativi esistenti con un efficace condivisione dei dati e delle informazioni di base.

Prima di attivare la messaggistica è fondamentale prevedere una fase iniziale per il corretto allineamento dell'anagrafica locale del sistema proposto con quella centrale.

I principali messaggi inviati dall'EMPI agli applicativi sono gli standard HL7 (v.2 e 3) ADT A28, ADT A29, ADT A31 e ADT A40, mentre i messaggi inviati dagli applicativi all'EMPI sono, secondo i casi, gli standard ADT A28 e ADT A31.

NOTA BENE:

Attualmente è in uso la versione HL7 2.5 CDA 2 con formattazione xml o pipe per tutti i messaggi anagrafici, ADT, ORM, OMG e MDM.

Si precisa tuttavia che l'infrastruttura di base del sistema informativo di AUSL della Romagna sta evolvendo sia verso la versione 3.0 sia verso lo standard FHIR (Fast Healthcare Interoperability Resources) secondo le tipologie di integrazione.

Come noto, infatti, lo standard FHIR, pur essendo costruito sugli standard, inclusi l'HL7 V2, l'HL7 V3, e il CDA (Clinical Document Architecture che è parte di HL7 V3), a differenza di essi, utilizza anche RESTful web services e open web technologies, che includono non solo l'XML (usato dai precedenti standard) ma anche i formati di dati JSON e RDF, estendendo pertanto le potenzialità di integrazione a un'ampia gamma di sistemi e dispositivi, quindi non solo sistemi EHR ma anche app mobile, dispositivi medici e wearable.

Pertanto, in fase di progetto esecutivo il fornitore sarà tenuto a adeguarsi alle specifiche fornite dall'Azienda USL in modo definitivo.

Per quanto riguarda l'inserimento di una nuova anagrafica (ADT 28), in alternativa, può essere richiesto che per l'inserimento in anagrafica il sistema debba richiamare una maschera dedicata dell'EMPI. In questo caso la ditta è tenuta a apportare le necessarie integrazioni.

Una particolare attenzione deve essere riservata ai messaggi di merge (ADT A40) in quanto al momento non è previsto che i sistemi dipartimentali possano inviare una richiesta di Merge verso EMPI azione a elevato rischio normalmente demandata a un intervento specializzato per il controllo di qualità del dato.

Per la ricezione del messaggio di Merge da EMPI, al contrario, è richiesto che l'anagrafe locale del sistema proposto conservi la storia degli eventi spostati su anagrafica Master. Ciò per garantire la ricostruzione della successione di eventi al fine di una corretta gestione di un eventuale messaggio di annullamento del Merge (A37 unmerge).

Il sistema proposto deve gestire le seguenti funzionalità:

1. Dare evidenza, con opportuna simbologia, della distinzione delle anagrafiche temporanee o ancora da certificare dall'EMPI; dare inoltre evidenza (ove richiesto) delle anagrafiche provenienti dall'EMPI.
2. Attivare meccanismi e accorgimenti per ridurre il rischio dell'inserimento di anagrafiche multiple.
3. Consentire la ricerca normalizzata delle anagrafiche per non considerare gli eventuali spazi o altri caratteri spuri nei campi nome e cognome.
4. Attendere il ritorno del codice EMPI, prima di inviare un qualsiasi evento verso il repository aziendale, nel caso di inserimento di un nuovo paziente.
5. Rispettare alcune regole finalizzate a ottimizzare la gestione dei campi fondamentali per l'anagrafica, per i quali sono state definite una serie di restrizioni (per es. caratteri non permessi) o applicazioni di regole per garantirne la corretta validazione anagrafica (nell'ambito dell'EMPI).

<u>Identificazione anagrafica</u>

Il sistema, nell'ambito dell'integrazione con l'anagrafe centrale EMPI, deve consentire la rilevazione dei dati anagrafici del paziente garantendo le seguenti funzionalità minime:

- Ricerca del paziente all'interno dell'anagrafe per una qualsiasi combinazione dei dati identificativi principali (cognome, nome, sesso, data e comune di nascita, codice fiscale).
- Inserimento di un paziente non presente in anagrafe, attraverso la registrazione dei seguenti dati obbligatori: cognome, nome, sesso, data e comune di nascita, codice fiscale, comune di residenza e cittadinanza. Ciò potrà avvenire indifferentemente da una maschera dell'applicativo o da una maschera messa a disposizione e appositamente richiamata dall'EMPI.

Va posta particolare attenzione sulla necessità di dover utilizzare anche l'anagrafica di un defunto.
Va inoltre prevista la gestione dei prodotti del concepimento abortiti, senza un nome ma con legami da porre con la mamma.
Va gestito, infine il caso di richiesta di anonimato nei casi previsti dalla normativa (pseudoanonimizzazione).

Le funzionalità di ricerca devono essere estremamente efficienti in modo da velocizzare l'attività e ridurre al minimo la necessità di inserimento di un nuovo paziente e la generazione di posizioni anagrafiche. È richiesta l'indicazione in offerta di parametri di valutazione per questo indicatore.

Il sistema deve garantire (come funzione configurabile) l'immodificabilità del dato anagrafico nei suoi campi fondamentali dopo l'invio della notifica di inserimento verso l'EMPI. La comunicazione a EMPI, infatti, rende quest'ultimo il proprietario dell'informazione: ciò comporta conseguentemente

la comunicazione del contenuto anagrafico a tutti i dipartimentali collegati, per cui ogni successiva modifica deve avvenire solo in maniera controllata e centralizzata.

Questo vincolo deve essere correttamente gestito dalla procedura senza limitare l'operatività degli utenti. In particolare il tema della corretta identificazione anagrafica va completato da un adeguato apparato di stampe (per es. per completare la documentazione sanitaria) e di messaggistica (per es. messaggi di warning), tra cui la stampa legata a una rettifica anagrafica corrispondente alla *move visit* (sposta evento).

Il fornitore deve dare adeguata evidenza dei meccanismi che il proprio sistema utilizza per garantire quanto qui richiesto.

Cap. Σ.3. – La cooperazione applicativa nell'AUSL della Romagna

In azienda è già attiva una complessa infrastruttura di cooperazione applicativa tra i diversi sistemi informatici presenti, che realizza tra di essi una completa integrazione definita utilizzando pienamente gli standard HL7 e DICOM secondo il modello di integrazione IHE. Il progetto prevede tra le sue componenti un portale per la gestione delle richieste di prestazioni da reparto (Order Entry), basato su un'anagrafe consolidata degli accessi (EMPI), e di un database condiviso per l'archiviazione dei referti, dei dati clinici, diagnostici ecc. (Clinical Data Repository).

Il fornitore dovrà inserirsi in modo ottimale nell'architettura presente collaborando con i diversi attori coinvolti.
Tuttle le integrazioni sono orchestrate da un middleware, tuttavia per alcuni flussi di integrazione ritenuti particolarmente critici sarà richiesta l'implementazione degli ack anche a livello di applicativo al fine di accertare in forma completa e sicura la corretta trasmissione della messaggistica tra inviante e destinatario.
Le specifiche di dettaglio saranno fornite in fase di progetto esecutivo.

Il sistema dovrà prevedere la ricezione della messaggistica relativa alle degenze e agli accessi di PS allo scopo di consentire il legame tra i referti prodotti e gli eventi clinici nell'ambito del quale vengono eseguite le richieste. Se ne riporta una descrizione sintetica:

- Ritorno al sistema di ADT e PS per le informazioni della richiesta (messaggi OMG) contenenti le informazioni relative allo stato della richiesta;
- Per il pronto soccorso si prevede l'uso di questo tipo di messaggi (tipo OMG) al fine di garantire anche il ritorno dell'erogato e il relativo calcolo ticket.

Recepire le informazioni relative ai degenti e agli accessi di PS (tramite le richieste informatizzate) implica la gestione delle informazioni relative a: codice unico evento (codice nosologico ADT o ID evento), codice anagrafico EMPI, codice braccialetto (se diverso dal codice ADT).

In aggiunta i messaggi da gestire e da recepire sono: *move visit* A45 (sposta evento) dal sistema di ADT/PS; A08 (aggiornamento del codice braccialetto, qualora diverso dal codice nosologico) dal sistema di ADT/PS/Presa in carico.

Al repository aziendale devono pervenire tramite messaggistica HL7: le richieste (messaggi OML); gli avanzamenti di stato della richiesta (accettata/eseguita/refertata/annullata/cancellata ecc. messaggi OMG); ritorno dei dati strutturati (messaggi ORU); i referti (messaggi MDM FSE compliant); eventuale rettifica e altre operazioni su referti redatti.

Le modalità di dettaglio saranno approfondite in fase di progetto esecutivo.

Codice percorso

La gestione del percorso in ambito di *specialistica ambulatoriale evoluta* o di *day service* prevede la completa presa in carico del paziente da parte del professionista che attiva il percorso, che pertanto provvede non solo alla prescrizione degli esami della propria o di altre discipline, ma anche alla prenotazione su agende riservate degli erogatori. Man mano che i referti sono da questi resi disponibili si costituisce il *dossier del percorso* che è reso visibile al richiedente e, secondo i casi, a uno o più attori del percorso incluso eventualmente l'MMG.

La gestione dei percorsi è resa possibile dall'associazione, in tutti i gestionali, di un *codice percorso* (diverso per ogni tipo di percorso) agli eventi associati.

Il sistema fornito, pertanto, nella sua qualità di erogatore, deve acquisire e gestire nel suo processo interno le informazioni relative al percorso (un codice identificativo e i suoi relativi attributi es. data apertura, medico che apre il percorso ecc.) che sarà veicolato attraverso i sistemi di order entry (incluso il sistema CUP).

Il gestionale deve ove richiesto inviare tramite messaggi (OMG) lo stato di avanzamento delle prestazioni associate al percorso al fine di permetterne il corretto monitoraggio dagli attori del percorso.

Il codice percorso (associato all'ID paziente) deve essere inserito nella parte documentale inviata al repository (messaggi MdM), qualora i referti redatti siano eseguiti su prestazioni pervenute con il codice percorso associato.

Il codice percorso, infine, deve essere esportato nel tracciato prodotto a fini statistici.

Cap. Σ.4. – Gestione consensi e oscuramento

L'Azienda si è dotata di un sistema centralizzato di gestione dei consensi al trattamento dei dati in varie tipologie e declinazioni (Privacy manager).

Il sistema deve interagire in modo bidirezionale con il Privacy Manager acquisendo lo stato dei consensi e restituendo eventuali inserimenti, revoche e modifiche dello stato.

Le modalità di interscambio dati con il Privacy Manager possono essere indifferentemente web service di tipo REST con payload custom JSON e xml/json HL7 FHIR, sia con code MQTT con payload json custom o FHIR.

Si precisa che la visibilità dei dati per effetto della contitolarità tra le due aziende AUSL della Romagna e IRST, secondo le modalità meglio precisate in altre parti del capitolato, può avvenire eslcusivamente previa verifica dell'esistenza del consenso positivo a tale condivisione da verificare nel Privacy Manager.

Gli applicativi, tipicamente attraverso il RIS, devono consentire la gestione di tutti i consensi necessari per l'attività ordinaria, dal consenso alla gestione dei dati nel dossier sanitario o per la costituzione di registri o l'adesione a reti di patologia, al consenso informato per i trattamenti, sino al consenso per il trattamento dati nel circuito SOLE/FSE, nonché alla gestione della volontà di oscuramento. In merito a quest'ultimo elemento, infatti, la normativa prevede l'obbligo di ricordare all'assistito, per ogni singolo evento, la facoltà di poterlo oscurare ovvero di impedirne la trasmissione al Dossier Aziendale e/o al FSE (sono due azioni separate). Il sistema deve pertanto rendere ergonomico e agevole il rispetto di questa indicazione, favorendone l'inserimento e tracciando le azioni di oscuramento/deoscuramento e storicizzandole.

Il sistema deve distinguere, in modo configurabile, il comportamento da adottare in caso di mancata manifestazione di tale volontà: per es. se l'operatore non chiede nulla all'utente rispetto alla sua volontà di oscurare quel particolare evento il sistema potrebbe inviare l'evento al Dossier e/o al FSE (configurazione in modalità silenzio-assenso) oppure inibire l'invio (configurazione in modalità silenzio-diniego).

Per alcune tipologie di esami o per alcune discipline può essere impostato il cosiddetto "oscuramento d'ufficio" (o da regolamento). In questo caso il referto deve essere inviato nello stato di "oscurato" salvo indicazione contraria da parte del paziente. La procedura deve gestire correttamente anche questi casi.

Ulteriori precisazioni saranno fornite in fase di progetto esecutivo da ciascuna e possono variare nel tempo in base all'evoluzione della normativa e dei regolamenti aziendali.

Anche il tempo di latenza che precede l'invio di un evento al circuito SOLE deve essere liberamente configurabile dall'Amministratore di Sistema (es. invio dell'evento dopo 3 giorni dal suo verificarsi).

Cap. Σ.5. – Gestione centralizzata dizionari

E' in programma l'introduzione un sistema centralizzato per la gestione dei dizionari e delle codifiche le cui specifiche sono in fase di definizione e saranno fornite in fase di progetto esecutivo.

Tipicamente ogni nuova voce di dizionario (es. una nuova Unità Richiedente) sarà propagata ai dipartimentali mediante messaggistica di tipo HL7 MFN (Master files notification).

Il fornitore dovrà adeguarsi per una completa gestione di questo tipo di messaggistica.

Cap. Σ.6. – Gestione richieste

Il fornitore deve consentire la gestione delle richieste di esami con due distinte modalità:

1. Fornitura* di un apposito modulo di richiesta che possa essere richiamato sia in modo indipendente in modalità web, sia in modo integrato e contestualizzato da altro applicativo (sia invocato dall'applicativo con passaggio di contesto, sia direttamente all'interno di un frame dell'applicativo chiamante). Tale modulo deve essere in grado di gestire sia le richieste per pazienti degenti (interni) con le relative informazioni (codice EMPI e informazioni paziente, medico richiedente e tipologia, reparto richiedente, reparto di ricovero, sala o blocco operatorio ecc.), sia le richieste per pazienti ambulatoriali (esterni) con le relative informazioni (codice EMPI e informazioni paziente, ambulatorio richiedente, codice ricetta, dati obbligatori per i flussi regionali, percorso day service ecc.). **(*Opzionale come espressamente indicato anche nella scheda d'offerta)**
2. Utilizzando programmi di richiesta Order Entry di terze parti già presenti in Azienda o eventualmente adottati in futuro.

Un esempio di messaggistica HL7 inerente a questo tipo di integrazione è la seguente:

- OMG_O19: Inserimento Richiesta (segmenti MSH, PID, PV1, ORC, TQ1, OBR)
  In risposta al messaggio di inoltro della richiesta verrà inviato un messaggio ACK_O19 (segmenti MSH, MSA, ERR opzionale);
  OMG_O19: Cambia Stato Richiesta a "Pianificata" caratterizzata da ORC.5=HD;
  OMG_O19: Cambia Stato Richiesta a "Accettazione Diretta" caratterizzata da ORC.5=SC;
  OMG_O19: Cambia Stato Richiesta a "Erogata" caratterizzata da ORC.5=CM ;
  OMG_O19: Cambia Stato Richiesta a "Cancellata" caratterizzata da ORC.1=ORC.5=CA;
- MDM_T02: Inserimento Referto;
- MDM_T10: Sostituzione Referto;
- MDM_T11: Annullamento Referto.

Le specifiche complete della messaggistica saranno fornite in fase di progetto esecutivo.

Cap. Σ.7. – Gestione amministrativa (calcolo ticket, gestione prestazioni aggiuntive ecc.)

Il modello di cooperazione applicativa di AUSL della Romagna prevede un gestore unico centralizzato deputato al calcolo del ticket e all'emissione della documentazione per il pagamento secondo le specifiche PagoPA. Il sistema fornito dovrà dialogare in modo bidirezionale con tale gestore centralizzato che sarà parte integrante del sistema di prenotazione CUP. Si precisa che la gestione delle prestazioni aggiuntive con emissione di ricetta DEMA è invece demandata al sistema oggetto della fornitura.

Poiché il sistema è attualmente in fase di definizione, le relative specifiche di integrazione saranno fornite in fase di progetto esecutivo.

Cap. Σ.8. – L'infrastruttura di rete dell'AUSL della Romagna

L'infrastruttura di rete dell'azienda USL della Romagna è così caratterizzata:

1. Rete dati locali con aggregazione sui centri stella a 10Gb e distribuzione finale ad 1Gb.
2. Linee dati geografiche con 1Gb di banda e ridondanza di provider sulle sedi principali.

3. Datacenter regionali, in gestione a Lepida, come sede principale e di disaster recovery per i nuovi progetti.
4. Data center locali dedicati ai soli server non remotizzabili presso i 7 presidi ospedalieri.

Si riportano nelle seguenti due figure gli schemi di connettività da cui si evidenzia la ridondanza di collegamenti geografici acquisti dai due provider Lepida e TelecomItalia. La banda disponibile sulla connettività Lepida è di 1Gb e sarà adeguata a 10Gb in base al reale utilizzo di banda e comunque entro i tempi di realizzazione del progetto.

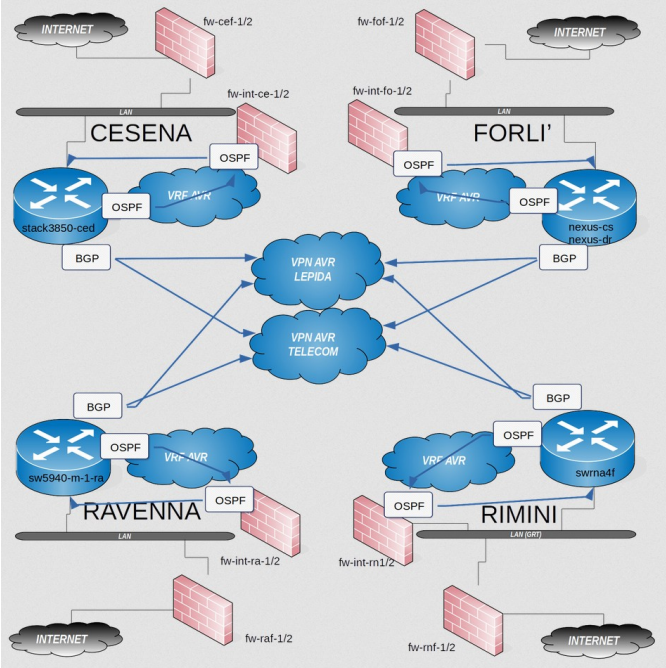


Fig. 1 – Principali sistemi di connettività interna ed esterna

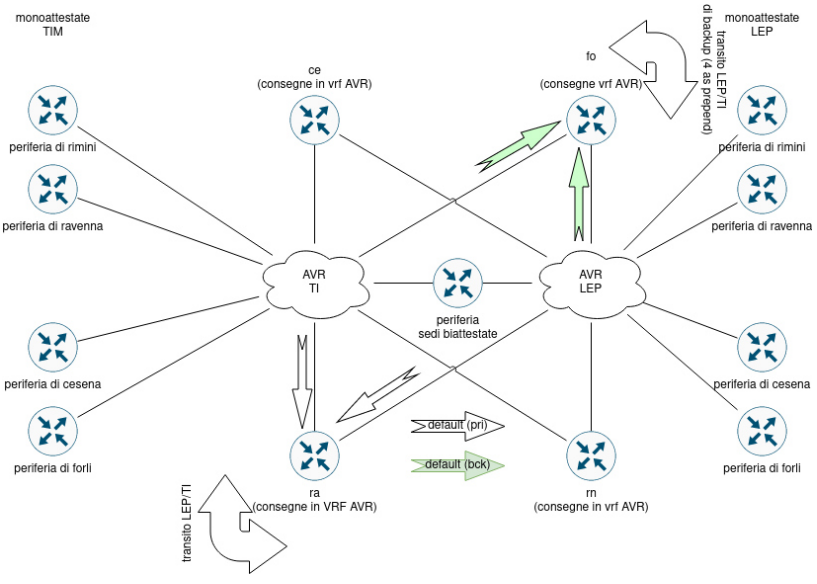


Fig. 2 – Schema delle ridondanze dei collegamenti geografici

Nella figura seguente è rappresentata l'architettura interna ai data center Lepida e la connettività di replica verso il data center di disaster recovery.

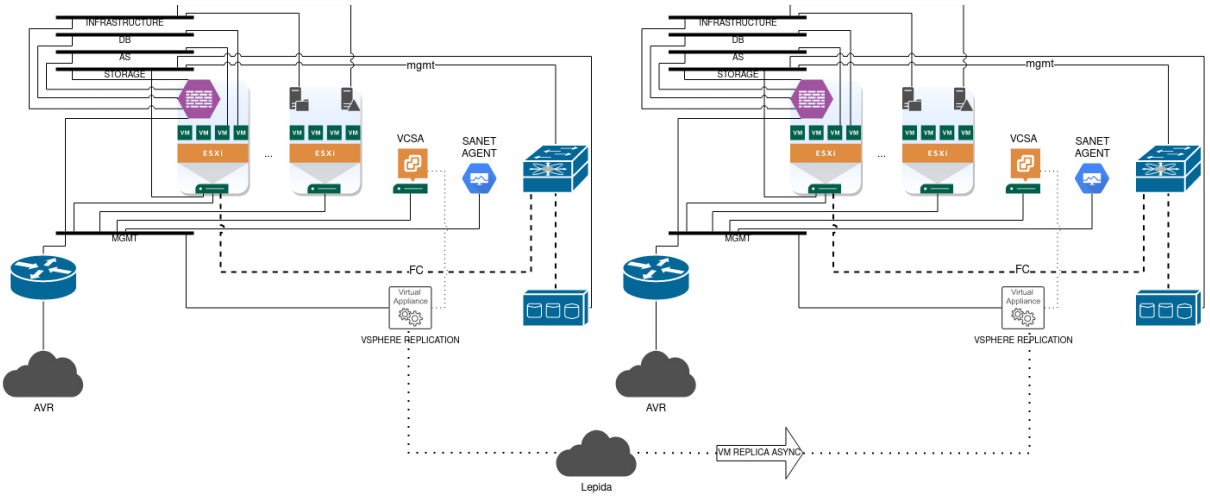


Fig. 3 – Rappresentazione dell'architettura server presso i data center Lepida

Il sistema offerto dovrà essere installato presso i data center di Lepida, secondo le modalità descritte.

La configurazione del networking, degli instradamenti e della gestione dei DNS sarà a carico dell'AUSL della Romagna e l'accesso dovrà essere garantito tramite l'utilizzo di credenziali personali dei singoli amministratori di sistema.

Sarà fornito dalla AUSL della Romagna il collegamento VPN per l'accesso da remoto finalizzato alle attività di manutenzione, l'accesso tramite VPN dovrà essere garantito tramite credenziali personali dei singoli amministratori di sistema (non sono ammesse pertanto configurazioni lan to lan).

Dovrà essere garantito l'invio dei log dell'attività degli amministratori di sistema (syslog) ai server di archiviazione centrali dell'Azienda USL.

In offerta dovranno essere specificati i requisiti di rete dati necessari al corretto funzionamento dell'intero sistema sia per la connessione fra server, sia per la connessione client/server.

Si precisa che, oltre agli oneri derivanti dall'utilizzo dei servizi di Lepida, tutte le licenze (non solo dei sistemi operativi e del RDMS, ma anche di ulteriori eventuali licenze necessarie per il corretto funzionamento del sistema offerto, sia lato server sia lato client, esempio: CAL, add-on, plug.in ecc.), sono interamente a carico del fornitore. Inoltre tutta l'infrastruttura software deve basarsi sulle ultime versioni disponibili e supportate dei prodotti (sistemi operativi, framework ecc.).

Su tutti i software di sistema e accessori, inoltre, dovranno essere installate e manutenute le eventuali patch almeno entro due mesi dalla data di rilascio da parte del produttore, compatibilmente con le esigenze di sicurezza e di compatibilità ambientale.

Eventuali situazioni di conflitto devono essere segnalate a AUSL e risolte congiuntamente.

Tutta l'infrastruttura software fornita dovrà essere costantemente aggiornata in conformità a quanto richiesto o derivante da direttive AGID o dalla giurisprudenza legata al GDPR, nonché dai regolamenti interni in vigore nelle aziende interessate.

Gli applicativi devono essere presentati ai client attraverso un unico punto di accesso: unico URL web su porta standard https, nascondendo quindi le eventuali complessità o molteplicità dei server e servizi presentati dietro al bilanciatore/reverse proxy.

Devono inoltre essere adottati meccanismi adeguati di disaccoppiamento (es. transazioni stateless o sessione condivisa tra gli application server) per evitare che venga persa la sessione dei client collegati a un particolare application server, qualora tale server dovesse perdere la disponibilità.

Architetture per la sicurezza

Il fornitore dovrà fornire il piano di backup del sistema offerto che dovrà consentire, per ogni funzionalità fornita, il ripristino delle informazioni fino all'ultimo “commit” eseguito. Il piano dovrà essere dettagliato e documentato in ogni sua parte, includendo anche il ripristino del sistema in produzione.

Il fornitore dovrà proporre una soluzione per la gestione del Disaster Recovery il cui dimensionamento e architettura dovranno garantire adeguati SLA in merito alle tempistiche e alle modalità di ripristino. In particolare dovranno essere previste le infrastrutture in un data center di Lepida diverso dal sito primario, per ospitare l’infrastruttura del sistema di DR secondo l’architettura proposta. L’offerta tecnica del fornitore dovrà indicare il dimensionamento complessivo delle macchine (CPU, RAM, disco) per soddisfare le esigenze del sistema di DR progettato.

Inoltre la soluzione proposta dovrà essere caratterizzata da un’architettura che garantisca la continuità operativa (business continuity) secondo un modello che sia compatibile con le infrastrutture disponibili in caso di rallentamento e/o malfunzionamento bloccante con interruzione del funzionamento di una parte qualsiasi del sistema primario.

Sistema di autenticazione

Il sistema offerto dovrà integrarsi con i meccanismi di autenticazione (Single Sign On) presenti nelle aziende interessate utilizzando le credenziali di autenticazione già assegnate agli operatori. In particolare il sistema di AUSL della Romagna utilizza il software open source Shibboleth.

Il sistema deve prevedere l’integrazione con altre procedure esterne senza richiedere una nuova autenticazione per l’utente. Tali integrazioni dovranno essere realizzate effettuando in modo sicuro il passaggio dei dati relativi all’utente collegato, che deve avvenire su canale cifrato e non essere replicabile.

Allegati: Specifiche Firma Digitale Remota Aruba (AUSL Romagna)

- ARSS Developer's Guide
- Credential_Proxy_Developers_Guide
- FirmaRemotaAruba

Si precisa che il provider ed anche conseguentemente la soluzione di firma remota utilizzate dalle Aziende potranno essere soggetti a cambiamenti nel corso della durata del contratto. E' onere da considerare nella redazione del progetto le conseguenti attività di adeguamento ed integrazione necessarie a seguito di tali eventuali cambiamenti.

ARSS_Developer's_Guide_2.7

ARSS
Developer's Guide

Versione 2.7
Revisione: 15 novembre 2019

Indice

Generalità

Scopo del documento

Il presente documento, rivolto agli sviluppatori di software, contiene informazioni necessarie per apprendere rapidamente l'uso del Web-Services **ARSS™** di Aruba PEC.
Questo documento non sostituisce, ma integra, la documentazione in formato HTML (javadoc) che accompagna il componente Server.

Requisiti per la lettura

Questo documento è stato scritto per sviluppatori software con una buona esperienza teorica e pratica sull'utilizzo di Web-Services basati su protocollo SOAP. Si presuppone, inoltre, che il lettore abbia familiarità coi concetti di crittografia a chiave pubblica, hashing, encryption e firma digitale, certificati X.509, certification authorities, buste crittografiche, pkcs#11.

Definizioni e acronimi

API Application Programming Interface
CA Certification Authority
CAPI CryptoAPI
CNIPA Centro Nazionale per l'Informatica nella Pubblica Amministrazione (attualmente DigitPA)
CSR Certificate Signing Request
DER Distinguished Encoding Rules
DES Data Encryption Standard
DN Distinguished Name
DPCM Decreto del Presidente del Consiglio dei Ministri
DPR Decreto del Presidente della Repubblica
GUI Graphical User Interface
HMAC Hash-based MAC
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
I/O Input/Output
LDAP Light-weight Directory Access Protocol
MAC Message Authentication Code
OCSP Online Certificate Status Protocol
PDF Portable Document Format
PEM Privacy-Enhanced Mail
PKCS Public Key Cryptographic Standard
PKI Public Key Infrastructure
RFUReserved For Future Use
RSA Rivest Shamir Adelman
SHA-1 Secure Hash Algorithm 1
SHA-256 Secure Hash Algorithm 256
SSL Secure Sockets Layer
TSDTime-Stamp Data
XML Extensible Markup Language Introduzione a ARSS

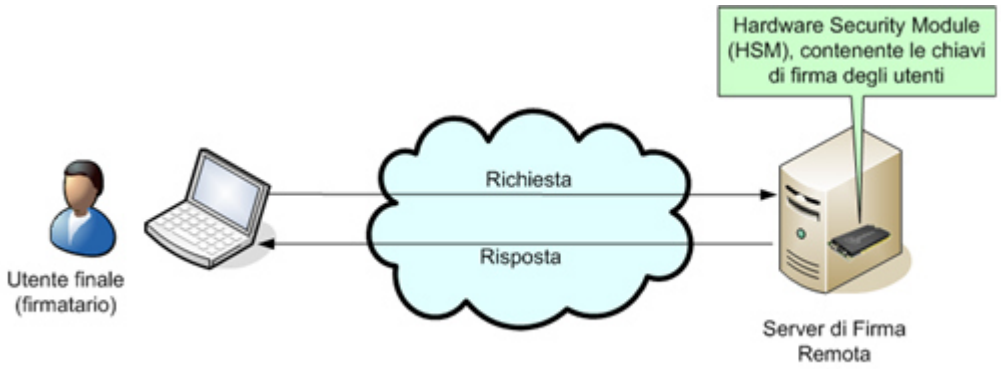
Introduzione ad ARSS

Firma Remota

La Firma Remota, è una modalità innovativa di firma digitale che, pur garantendo lo stesso grado di sicurezza e gli stessi effetti di legge della tradizionale firma digitale basata su smart card o token USB, rispetto a quest'ultima offre diversi vantaggi specifici:

- non richiede l'installazione di hardware e driver dedicati, pertanto riduce virtualmente a zero i relativi problemi di incompatibilità hw/sw, supporto tecnico, ecc.
- è sostanzialmente indipendente dall'ambiente operativo dell'utente (Windows, Mac, Linux, …)
- permette di generare firme digitali in ogni momento ed in ogni luogo.

In concreto, per "Firma Remota" si intende la firma digitale eseguita con una chiave privata non residente su un dispositivo personale dell'utente, quale ad es. una smartcard, bensì su un dispositivo remoto (normalmente un HSM – Hardware Security Module). I dati da firmare (o meglio la loro impronta) sono inviati all'HSM attraverso la rete, e la risposta ritorna all'utente sempre attraverso la rete, come illustrato in modo semplificato nella seguente figura:



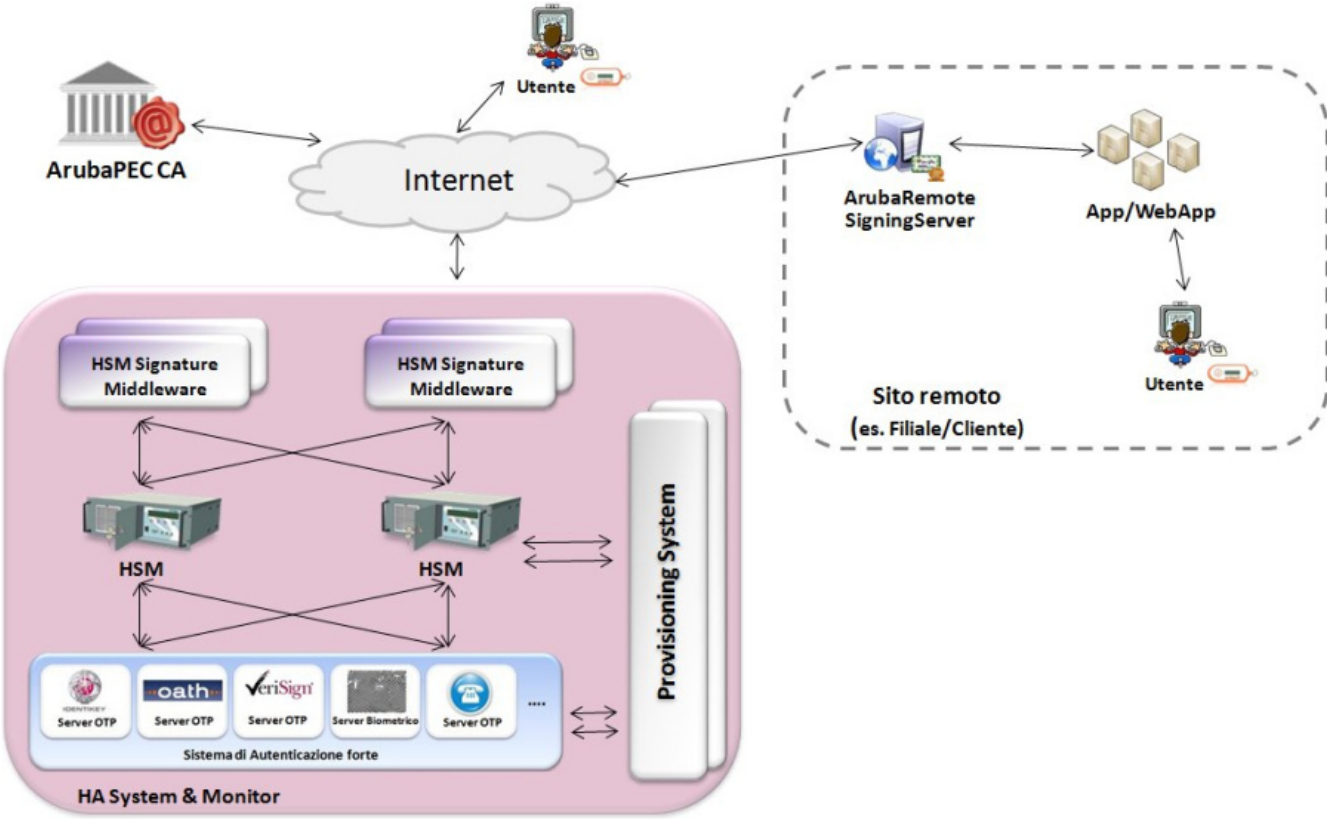
Il dialogo tra l'applicazione client e il server di firma può essere così sintetizzato:

- l'utente si autentica nei confronti del server (vedere più avanti per i dettagli);
- il client richiede la firma digitale, inviando al server il digest (hash) del documento;
- il server calcola la firma e la restituisce al client, dove viene salvata.

Lo stesso tipo di soluzione è utilizzabile – con gli stessi vantaggi – sia per apporre firme su singoli documenti, in modo interattivo, sia per firmare in modo "massivo" una serie di documenti (es. intera cartella)

Architettura del servizio di Firma Remota

La seguente figura mostra l'architettura di riferimento del sistema di firma remota implementato da Aruba PEC.



Il sistema è costituito dalle seguenti componenti architetturali:

- **HSM CoSign** – HSM all'interno dei quali sono generate e custodite le chiavi e i certificati digitali
- **HSM Signature Middleware** – componente software di gestione di tutte le richieste da e verso gli HSM
- **Sistema di Autenticazione forte** – il Sistema di Autenticazione forte è formato da varie compo¬nenti che possono essere interfacciate agli HSM CoSign e che consentono un'autenticazione forte del Titolare per lo sblocco delle varie operazioni di firma sull'HSM stesso

- **Provisioning System** - componente software che gestisce (orchestra) il dialogo con gli HSM, con il Sistema di Autenticazione forte e con la Certification Authority per la corretta attivazione del servizio di firma remota

- **HA System & Monitor** - componente software, trasversale all'intero sistema di firma remota, formato da vari moduli che gestiscono il monitoraggio del Sistema e che implementano tutte le funzioni necessarie a garantire l'alta disponibilità del servizio (fault-tolerance)

- **Aruba Remote Signing Server (ARSS)** – componente software che consente la remotizzazione delle funzionalità di firma digitale proprie dell'HSM. ARSS consente quindi una semplice integrazione delle funzionalità di firma digitale in applicazioni e sistemi "remoti", ossia erogati da siti di produzione diversi da quello dove risiede il sistema di firma remota.

ARSS

Aruba Remote Signing Server è il componente software che permette una semplice integrazione delle applicazioni e dei sistemi con il servizio di firma remota di Aruba PEC.
Nel caso di applicazioni (già in uso o future) ospitate in infrastrutture IT (siti di produzione) differenti da quella dove risiede il sistema di Firma Remota, ARSS provvederà a dialogare, su canale sicuro di comunicazione (HTTPS) con mutua autenticazione, con il sistema di firma remota di Aruba PEC, esponendo verso le applicazioni in questione le funzionalità di firma digitale. Tali funzionalità sono rese disponibili tramite una semplice interfaccia Web Services (SOAP) e includono (elenco non esaustivo):

- firma di un documento in formato CADES-BES e CADES-T (secondo quanto previsto dalla Deliberazione 45/2009 e successiva Determina 69/2010)
- firma di un documento in formato PADES-Basic, PADES-BES e PADES-T (secondo quanto previsto dalla Deliberazione 45/2009 e successiva Determina 69/2010)
- firma di un documento in formato XADES-BES e XADES-T (secondo quanto previsto dalla Deliberazione 45/2009 e successiva Determina 69/2010)
- firma di tutti i documenti contenuti in una cartella specificata, nei formati sopraindicati
- firma di un hash (impronta), nel caso in cui le applicazioni provvedano autonomamente alla creazione del documento firmato digitalmente tramite le API e/o i servizi messi a disposizione del fornitore
- marcatura temporale di un documento

La seguente tabella fornisce una rassegna delle funzionalità offerte da ARSS:

Messaggio/Metodo	**Funzionalità**	**Paragrafo**
addpkcs7sign	Aggiunta Firma Congiunta (Firma Parallela)	5.4
closesession	Chiusura di una sessione di Firma Massiva	5.2
getVersion	Restituzione versione ARSS	10.5
listCert	Restituzione elenco dei certificati associati ad un utente remoto tramite credenziali utente	10.1
listCertAuth	Restituzione elenco dei certificati associati ad un utente remoto tramite credenziali amministrative	10.2
authMethods	Restituisce la lista dei metodi di autenticazione di un utente	10.3
listprocess	Restituzione della lista dei processi attivi di firma per utente	10.4
m7m	Metodo Rimosso, Generazione Marche Temporali in formato **m7m**	8.4
opensession	Apertura di una sessione di Firma Massiva	5.2
Ping	Controllo della raggiungibilità del servizio remoto	10.6
pdfsignatureV2	Apposizione di una firma PAdES (-BES, -T) ad un file **pdf**	6
pdfsignature	Metodo Deprecato, vedi pdfsignatureV2	N/A
pkcs7signV2	Apposizione di una firma CAdES (-BES, -T) ad un file generico	5.1
pkcs7sign	Metodo Deprecato, vedi pkcs7signV2	N/A
pkcs7signhash	Apposizione di una firma CAdES (-BES, -T) ad un impronta	5.5
Tsd	Generazione Marche Temporali in formato **tsd**	8.3
Tsr	Generazione Marche Temporali in formato **tsr**	8.2

Verify	Metodo Deprecato, non disponibile una nuova versiohne	N/A
xmlsignature	Apposizione di una firma XAdES (-BES, -T) ad un file **xml**	7.2
signhash	Apposizione di una firma pkcs1 all'impronta di un file	4.1
verifyOtp	Autenticazione mediante credenziale OTP	10.7
sendCredential	Invio della credenziale OTP su cellulare via ARUBACALL o SMS	10.8
retriveCredential	Recupera informazioni complementari di un0autenticzione estesa	10.9
pkcs7signhash_multiple	Apposizione multipla di firme CAdES (-BES, -T) ad un impronta	5.2.1
pdfsignatureV2_multiple	Apposizione multipla di firme PAdES (-BES, -T) ad un file **pdf**	5.2.1
xmlsignature_multiple	Apposizione multipla di firme XAdES (-BES, -T) ad un file **xml**	5.2.1
pkcs7signV2_multiple	Apposizione multipla di firme AdES (-BES, -T) ad un file generico	5.2.1
encryptedEnvelope	Crittazione di un file	10.10

La seguente tabella riassume gli standard e gli algoritmi supportati da ARSS.

Algoritmi di crittografia a chiave pubblica	RSA
Algoritmi di hashing	SHA-1, SHA256
Algoritmi di firma digitale	SHA1withRSA e SHA256withRSA con padding PKCS#1 1.5
Formato buste crittografiche	CADES-BES e CADES-T (ETSI TS 101 733)
Formato buste PDF	PADES-Basic, PDES-BES, PADES-T (ETSI TS 102 778)
Formato buste XML	XADES-BES e XADES-T (ETSI TS 101 903)
Formato marche temporali (time stamp tokens)	Rfc 3161
Protocollo di accesso al servizio di timestamping	Rfc 3161
Formato delle buste con marcatura temporale	Rfc 5544 e m7m

Il componente ARSS ha superato con successo i test di interoperabilità con tutti i certificatori accreditati DigitPA relativi alla produzione di firme in formato CAdES, PAdES e XAdES.

Gestione deleghe

A partire dalla versione 1.9 di ARSS sono stati introdotti dei metodi che consentono l'integrazione con l'infrastruttura ASB (Aruba Security Box) .Tra le modifiche più importati ci sono i metodi di firma automatica attraverso **delega** che hanno portato alla modifica della classe Auth implementata da ARSS.
La modifica estende il set di property della classe Auth in modo da poter sottoporre, in fase di sottoscrizione, le credenziali dell'utente delegato dal Titolare della chiave di firma.

Metodi di autenticazione estesi

Dalla versione 1.9 sono stati introdotti metodi di autenticazione forte che permettono ad arss di autenticare gli utenti con sistemi differenti dal OTP.
L'oggetto auth e stato modificato per accettare anche parametri di autenticazione estesa complessi come dati binary.
In particolare il metodo di autenticazione CNS2 permetto con l'invio come blob di un pkcs7 firmato con un certificato CNS la firma di documenti senza dover specificare ne password utente ne otp.

Firma Digitale remota in pkcs1

Firma Digitale di un'impronta secondo standard pkcs1

Se s'intende effettuare la firma dell'impronta di un documento (o di una sequenza generica di byte) basata sullo statndard pkcs1 è sufficiente utilizzare il metodo signhash().
Tale metodo è così definito:

```
public SignHashReturn signhash(SignHashRequest request)
```

Questo metodo viene normalmente utilizzato per i seguenti scopi:

- Generazione di firme digitali "detached" cioè separate dal file firmato

INPUT
request - Oggetto SignHashRequest contenente i seguenti parametri per la richiesta di firma
Identity = Oggetto Auth contenente i dati per l'autenticazione del firmatario
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
ext_authtype = Enum che consente di utilizzare il metodo di autenticazione esteso.
CNS2 -> Indica l'autenticazione tramite carta CNS
ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
ext_auth_blobvalue = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
ext_auth_value = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
delegated_user = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe.
In questo caso non deve essere specificata la password del Titolare della chiave di Firma
delegated_password = Stringa che contiene la password dell'utente delegato
delegated_domain = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)

certID = Identificativo univoco del certificato da utilizzare per la firma. L'elenco degli ID è recuperabile dalla funzione listCert esposta in seguito.
In caso non si necessiti di un certificato specifico si può specificare AS0 al fine di ottenere il bilanciamento delle richieste, tale metodo però presume che il certificato stesso non concorra al calcolo dell'HASH (non valido per firma CADES).
hash = Array di byte contenente l'impronta del documento da firmare.
hashtype = Stringa contenente il nome dell'algoritmo di hash usato per il calcolo dell'impronta (attualmente disponibile solo "SHA256").
session_id = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla.
Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
requirecert = booleano che indica la richiesta della presenza del certificato del firmatario nella risposta.
OUTPUT
Oggetto SignHashReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di firma
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignHashReturn.Description.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di firma
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0007" -> Profilo Di firma PDF non valido
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
signature = Array di byte contenente il risultato del processo di firma.
cert = Array di byte contenente certificato del destinatario se richiesto vedi parametro requirecert.
certID = Identificativo del certificato del firmatario.

Firma Digitale remota in standard CAdES (p7m)

Firma Digitale

Il metodo da utilizzare per generare una firma digitale di un singolo file è pkcs7signV2().
Il metodo è così definito:

```
public SignReturnV2 pkcs7signV2 (SignRequestV2 request, boolean detached, boolean returnder)
```

INPUT
request - Oggetto SignRequestV2 contenente i seguenti parametri per la richiesta di firma
Transport = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da firmare
FILENAME -> Nel caso in cui si stia passando in input il percorso del file da firmare
DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da firmare
STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)

Binaryinput = Byte array contenente il documento da firmare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
SrcName = Stringa contenenti il path del file o della cartella contenente i files da firmare.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
Stream = DataHandler per caricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS
DstName = Stringa contenente il path del file o della cartella di destinazione dei file firmati.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS
requiredmark = Boolean utilizzato per pilotare la generazione di una busta CAdES-T: firma digitale con apposizione contestuale della marca temporale.
TRUE -> Nel caso in cui s'intenda generare un CAdES-T
FALSE -> In caso contrario
NOTA: Nel caso in cui s'intenda effettuare la marca temporale in formato CAdES-T, il server ARSS dovrà essere preventivamente configurato con un account di marcatura temporale valido.
signingTime = Stringa utilizzata per la valorizzazione dell'attributo signingtime (OID 1.2.840.113549.1.9.5) da inserire nella busta CAdES.
dd/MM/yyyy HH:mm:ss -> Nel caso in cui s'intenda specificare esplicitamente la data di firma
NULL -> Nel caso in cui s'intenda demandare ad ARSS la valorizzazione dell'attributo.
NOTA: ARSS inserirà la data di sistema del Server su cui è in esecuzione.
Session_id = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla.
Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
Identity = Oggetto Auth contenente i dati per l'autenticazione del firmatario
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
ext_authtype = Enum che consente di utilizzare il metodo di autenticazione esteso.
CNS2 -> Indica l'autenticazione tramite carta CNS
ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
ext_auth_blobvalue = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
ext_auth_value = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
delegated_user = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe.
In questo caso non deve essere specificata la password del Titolare della chiave di Firma
delegated_password = Stringa che contiene la password dell'utente delegato
delegated_domain = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
tsa_identity = Oggetto TSAAuth opzionale, contenente i dati per l'autenticazione Al Server TSS. Se non specificato il server userà l'account di default se configurato.
user = Stringa contenente la user dell'utenza di marcatura temporale
password = Stringa contenente la password dell'utenza di marcatura temporale

Notitymail = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di firma.
Notity_id = Stringa contenente l'id che l'utente intende associare alla sessione di firma per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
Nel caso di firma di singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
CertID = RFU. Valorizzare con AS0
profile = RFU. Impostare a NULL

detached – Boolean utilizzato per la generazione di una busta CAdES detached, cioè senza il documento originale.
true = richiesta di generazione di firma CAdES senza non contenente il documento originale (detached)
false = richiesta di generazione di firma CAdES contenente il documento originale

returnder – Boolean utilizzato per la generazione di una busta CAdES in formato DER
true = richiesta di generazione in formato DER
false = richiesta in formato BER (è il valore di default)

signatureLevel – Livello di firma richiesto. (Valore Ammesso: LT)

OUTPUT
Oggetto SignReturnV2 contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di firma
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignReturnV2.description e SignReturnV2.return_code.
Binaryoutput = Byte array contenente il file firmato
Stream = DataHandler per scaricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

DstPath = Stringa contenente il percorso specificato all'interno del parametro SignRequestV2.DstName in fase di richiesta.
Nel caso in cui sia il file specificato in SignRequestV2.DstName già esista la property SignReturnV2.DstPath contiene il percorso al file effettivamente generato da ARSS.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di firma
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0007" -> Profilo Di firma PDF non valido
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code

Firma Multipla

La Firma Multipla è quella modalità di firma utilizzata per sottoscrivere un insieme di file sottoponendo una sola volta le credenziali di autenticazione.
Dalla versione 1.9 (vedi capitolo 10) è possibile usare anche i metodi di firma multipla che di fatto automatizzano quanto descritto in questo capitolo e lo ottimizzano.
Per esigenze particolari è comunque utile tulizzare i motodi fi firma multiple separati.
Per effettuare la firma digitale di un insieme di file in formato p7m è possibile utilizzare due diversi approcci:

- Firmare una cartella visibile dal Server ARSS nella quale sono posizionati tutti i file da firmare
- Firmare in modo reiterato i singoli file che costituiscono l'insieme di input
- Dalla versione 1.9 si posso usare anche i metodi di firma multipla che di fatto automatizzano quanto descirtto con il metodo due. (Vedi Capitolo **Errore. L'origine riferimento non è stata trovata.**)

Entrambe le modalità consentono di effettuare la sottoscrizione dei file attraverso l'inserimento di un singolo codice OTP senza però diminuire per questo il livello di sicurezza del processo di firma.
Firma di una cartella
In questo caso è sufficiente utilizzare il metodo pkcs7signV2() già descritto al paragrafo *Firma Digitale* con i seguenti accorgimenti:

1. Impostare la property SignRequestV2.Transport a DIRECTORYNAME
2. Valorizzare la property SignRequestV2.SrcName con il percorso della cartella di destinazione che conterrà i file firmati
3. Valorizzare opportunamente le property SignRequestV2.Notity_id e SignRequestV2.Notitymail per ricevere un messaggio di riscontro al completamento dell'operazione di firma della cartella

Firma reiterata dei singoli file
In questo caso è necessario utilizzare una sequenza di chiamate a metodi così strutturata:
opensession(session_ID)
pkcs7signV2(file_#1, session_ID)
pkcs7signV2(file_#2, session_ID)
...
pkcs7signV2(file_#n-1, session_ID)
pkcs7signV2(file_#n, session_ID)
closesession(session_ID)
Segue il dettaglio di invocazione di ciascuna tipologia di metodo:

```
public String opensession(Auth identity)
```

Metodo utilizzato per l'apertura della sessione di firma multipla

INPUT
identity - Oggetto Auth contenente i seguenti parametri per la richiesta di firma
TypeHSM = "COSIGN"
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
OUTPUT
Stringa contenente l'id della sessione appena aperta nel caso in cui l'operazione sia andata a buon fine.
Nel caso in cui l'operazione vada in errore viene ritornata una delle seguenti stringhe:
"KO-0001" -> Errore generico
"KO-0003" -> Errore in fase di verifica delle credenziali
"KO-0004" -> Errore nel PIN
"KO-0009" -> Credenziali di delega non valide
"KO-0010" -> Lo stato dell'utente non è valido (es. utente sospeso)

```
public SignReturnV2 pkcs7signV2 (SignRequestV2 request, boolean detached)
```

Il metodo pkcs7signV2 verrà invocato **n** volte dove **n** sono il numero di file da firmare**.**
INPUT
I dettagli di invocazione di tale metodo seguono quanto già descritto al paragrafo *Firma Digitale* con le seguenti modifiche contestualizzate per il processo di firma multipla:

1. Impostare la property SignRequestV2.Session_id con l'id di sessione ritornato in output dalla chiamata preliminare alla opensession
2. Impostare la property SignRequestV2.Identity con i dati di autenticazione del firmatario

NOTA: In questo caso non è necessario valorizzare la property SignRequestV2.Identity.OtpPWD dato che l'autenticazione dell'utente avviene attraverso la terna (User, UserPWD, SessionID)

1. Passare in input il singolo file su cui s'intende apporre la firma

OUTPUT
L'output restituito da ogni chiamata seguirà quanto già indicato nella corrispondente sezione del paragrafo *Firma Digitale*

```
public String closesession(Auth identity, String sessionid)
```

Metodo utilizzato per la chiusura della sessione di firma multipla.
INPUT
identity - Oggetto Auth contenente i seguenti parametri precedentemente specificati in fase di apertura della sessione
TypeHSM = "COSIGN"
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Non necessario perché l'autenticazione è basata sul sessionID.
ext_authtype = Enum che consente di utilizzare il metodo di autenticazione esteso.
CNS2 -> Indica l'autenticazione tramite carta CNS
ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
ext_auth_blobvalue = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
ext_auth_value = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
delegated_user = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe.
In questo caso non deve essere specificata la password del Titolare della chiave di Firma
delegated_password = Stringa che contiene la password dell'utente delegato
delegated_domain = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
Sessionid – Stringa contenente l'id di sessione che s'intende terminare.
Tipicamente contiene la stessa stringa ritornata dalla chiamata opensession iniziale corrispondente alla sessione di firma multipla
OUTPUT
Stringa contenente l'esito dell'operazione:
"OK" -> Errore generico
"KO-0001" -> Errore generico

Metodi di firma Multipla

I metodi di firma multipla ricalcano nella forma e nella sostanza gli analoghi metodi utilizzati per la firma singola. L'unica differenza è nel fatto che accettano in ingresso **array di richieste** e restituiscono un **array di risposte**.

Di seguito il dettaglio di invocazione di ciascuna tipologia di metodo:

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="43f86cc9-e731-427c-89b0-6a548e44b704"><ac:plain-text-body><![CDATA[public SignReturnV2_multiple pkcs7signhash_multiple (Auth identity, SignRequestV2[] requestList, boolean countersignature)]]></ac:plain-text-body></ac:structured-macro> <ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="a830832c-a42e-445a-978e-d8779ec1f519"><ac:plain-text-body><![CDATA[public SignReturnV2_multiple pkcs7signV2_multiple (Auth identity, SignRequestV2[] requestList, boolean detached)]]></ac:plain-text-body></ac:structured-macro> <ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="bf4f58ef-55e4-46f2-811f-2e8319e1f8d3"><ac:plain-text-body><![CDATA[public SignReturnV2_multiple pdfsignatureV2_multiple (Auth identity, SignRequestV2[] requestList, PdfSignApparence apparence, PDFProfile pdfprofile)]]></ac:plain-text-body></ac:structured-macro> <ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="7c578146-852c-42f3-be28-85a1db61dc16"><ac:plain-text-body><![CDATA[public SignReturnV2_multiple xmlsignature_multiple (Auth identity, SignRequestV2[] requestList, XmlSignatureParameter parameter)	]]></ac:plain-text-body></ac:structured-macro>

Con l'intento di non dilungarsi nell'ulteriore nella descrizione di ciascun metodo nel seguito verrà fornito un esempio di invocazione del metodo per firma multipla Cades (pkcs7signV2_multiple).
INPUT
identity = Oggetto Auth contenente i dati per l'autenticazione dell'utente che sarà usato per elaborare tutte le richieste di firma contenute nell'array (cfr Paragrafo 5.1)

SignRequestV2[]

= Array di oggetti SignRequestV2 contenenti gli stessi parametri per la richiesta di firmma descritti al paragrafo 5.1
detached = Boolean utilizzato per l'indicazione di generazione di buste Cades detached (cfr Paragrafo 5.1)
OUTPUT
SignReturnV2_multiple = Oggetto complesso composto dai seguenti campi
private String status = Esito globale dell'operazione di firma multipla.
private String Description = Descrizione aggiuntiva dello stato globale
private String return_code = Codice di ritorno globale

private SignReturnV2[] return_signs

= Array di oggetti SignReturnV2 relativo alle richieste

Firma Automatica

Per effettuare una firma automatica è sufficiente utilizzare il metodo pkcs7signV2() passando in input le credenziali di autenticazione di un'utenza di firma automatica.
NOTA: Per riuscire ad effettuare una operazione di firma automatica l'utente deve aver preventivamente abilitato la relativa utenza attraverso le pagine di attivazione (https://attivazioni.firma-remota.it/asmonitor/)

public SignReturnV2 pkcs7signV2 (SignRequestV2 request, boolean detached, Boolean returnder)

INPUT
I dettagli di invocazione del metodo pkcs7signV2()seguono le indicazioni già riportate nella relativa sezione del paragrafo *Firma Digitale* con le seguenti modifiche contestualizzate per il processo di firma automatica:

1. Impostare la property SignRequestV2.Identity.TypeOtpAuth con la stringa identificativa del dominio di firma automatica.

NOTA: Questa stringa viene indicata in fase di installazione dell'ARSS

1. Impostare la property SignRequestV2.Identity.OtpPWD con la stringa identificativa delle transazioni di firma automatica

NOTA: Questa stringa statica viene definita in fase di installazione del server ARSS ed è normalmente nota all'amministratore di rete dell'infrastruttura IT su cui opera l'utente.
OUTPUT

L'output restituito dalla chiamata a funzione segue quanto già indicato nella corrispondente sezione del paragrafo *Firma Digitale*

Firma Congiunta (firma parallela)

Per effettuare una firma congiunta è sufficiente utilizzare il metodo addpkcs7sign()
Il metodo è così definito:

```
public SignReturnV2 addpkcs7sign (SignRequestV2 request, boolean detached)
```

INPUT
I dettagli di invocazione del metodo addpkcs7sign()seguono le stesse indicazioni già riportate per l'invocazione del metodo pkcs7signV2()(cfr paragrafo *Firma Digitale)* con le seguenti modifiche contestualizzate per il processo di firma parallela:

1. Il metodo accetta in input solo buste crittografiche in formato CAdES o PKCS#7
2. Il metodo può prendere in input anche cartelle contenenti esclusivamente file di cui al punto 1
3. Il parametro detached non deve essere valorizzato.

OUTPUT
L'output restituito dalla chiamata a funzione segue quanto già indicato per il metodo pkcs7signV2()all'interno della corrispondente sezione del paragrafo *Firma Digitale* con le seguenti modifiche contestualizzate per il processo di firma parallela:

1. Nel caso in cui siano stati passati in input dei file con formato diverso da quello CAdES o PKCS#7 l'errore restituito all'interno della property SignReturnV2.Return_code è "0001" (Errore generico nel processo di firma)

Firma Digitale di un'impronta

Se s'intende effettuare la firma dell'impronta di un documento (o di una sequenza generica di byte) è sufficiente utilizzare il metodo pkcs7signhash().
Tale metodo è così definito:

```
public SignReturnV2 pkcs7signhash(SignRequestV2 request, boolean countersignature, boolean excludeSigningTime)
```

Questo metodo viene normalmente utilizzato per i seguenti scopi:

- Apposizione di controfirme
- Generazione di firme digitali "detached" cioè separate dal file firmato

INPUT
countersignature – Boolean utilizzato per la generazione di una busta CAdES detached dalla quale è possibile estrarre gli oggetti necessari alla costruzione raw di una controfirma.
false = richiesta di generazione di firma CAdES detached per costruzione busta crittografica di firma standard
true = richiesta di generazione di firma CAdES detached per la costruzione di una controfirma
Maggiori dettagli in merito alla procedura per l'implementazione della controfirma sono nella sezione OUTPUT del presente paragrafo
excludeSigningTime– Boolean utilizzato per l'esclusione del signed attribute (signing-time) di una busta CAdES detached. Utile nel caso di costruzione di una busta PAdES a partire dall'impronta
false = inclusione del signed attribute nella busta della firma CAdES detached
true = esclusione del signed attribute nella busta della firma CAdES detached
request - Oggetto SignRequestV2 valorizzato conformemente alle indicazioni contenute all'interno della relativa sezione del paragrafo *Firma Digitale* con le seguenti modifiche contestualizzate per il processo di firma di un'impronta.

1. La property Transport deve essere valorizzata a BYNARYNET
2. Le property SrcName e DstName non devono essere utilizzate
3. La property Binaryinput deve contenere i 32 bytes che rappresentano l'impronta SHA-256 dell'oggetto da sottoscrivere (tipicamente l'impronta del documento che s'intende firmare)

OUTPUT
Oggetto SignReturnV2 contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di firma
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignReturnV2.description e SignReturnV2.return_code.
Binaryoutput = Byte array contenente la busta CAdES "detached" relativa all'hash specificato in fase di chiamata.
NOTA: La busta contiene all'interno del signedattribute SignerInfo.messagedigest i 32 bytes specificati all'interno dell'array Binaryinput

Stream = DataHandler per scaricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.
DstPath = Non utilizzato
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di firma
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code
Controfirma
Nel caso in cui il boolean countersignature sia stato impostato a TRUE è possibile utilizzare l'output restituito dal metodo pkcs7sighash() per la derivazione degli oggetti necessari alla costruzione di una controfirma in base alle seguenti indicazioni:
Indicando con 'Busta_A' il file CAdES al quale si vuole aggiungere la controfirma e con 'Busta_B' il file CAdES detached restituito dal comando pkcs7sighash(), di seguito è riportata una breve descrizione degli step da seguire per la costruzione di una controfirma

1. Individuare all'interno di Busta_A la struttura SignerInfo relativa alla firma da controfirmare
2. Estrarre i 128 bytes corrispondenti al valore del campo SingerInfo.signature (quindi no i tag ed i lenght octect)
3. Calcolare l'hash SHA256 dei 128 bytes ottenuti allo step precedente ed utilizzare i 32 bytes di output per la valorizzazione del byte array Binaryinput
4. Eseguire il metodo pkcs7signhash() settando a true il parametro countersignature
5. Dalla Busta_B risultante estrarre l'unica struttura SignerInfo presente ed utilizzarla come valore dell'unsigend-attribute countersignature della Busta_A
6. Dalla Busta_B estrarre l'unico certificato presente nel campo SignedData.certificates ed aggiungerlo al corrispondente campo della Busta_A
7. Aggiornare i lenght octets della Busta_A e salvare il risultato

Firma Digitale remota in standard PAdES (pdf)

Firma Digitale

Il metodo da utilizzare per generare una firma digitale di un singolo file è pdfsignatureV2()**.**
Dalla versione 1.14.0 il metodo include in automatico i font della firma all'interno del documento, preservando il formato PDF/A.
Tuttavia il formato PDF/A non supporta le immagini trasparenti, quindi se nel campo **Image o ImageBin viene valorizzato con un contenuto immagine trasparente,** il metodo produrrà un PDF non conforme al formato PDF/A.
Il metodo accetta in ingresso solo file pdf ed ha la seguente definizione:

```
public SignReturnV2 pdfsignatureV2(SignRequestV2 request, PdfSignApparence apparence, enum Pdfprofile, String password,
DictionarySignedAttributes dict_signed_attributes)
```

INPUT

- **request** - Oggetto SignRequestV2 contenente i seguenti parametri per la richiesta di firma
  - **Transport** = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
    - BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da firmare
    - FILENAME -> Nel caso in cui si stia passando in input il percorso del file da firmare
    - DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da firmare
    - STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)
  - **Binaryinput** = Byte array contenente il documento da firmare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
  - **SrcName** = Stringa contenenti il path del file o della cartella contenente i files da firmare. Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME. (NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS)
  - **Stream** = DataHandler per caricare file di grandi dimensioni in formato Stream. Utilizzato esclusivamente nel caso di Transport = STREAM. (NOTA: La soluzione e disponibile solo con client JAX-WS compliant.)
  - **DstName** = Stringa contenente il path del file o della cartella di destinazione dei file firmati. Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME. (NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS)
  - **requiredmark** = Boolean utilizzato per pilotare la generazione di una busta PAdES-T: firma digitale con apposizione

contestuale della marca temporale. (NOTA: Nel caso in cui s'intenda effettuare la marca temporale in formato PAdES-T, il server ARSS dovrà essere preventivamente configurato con un account di marcatura temporale valido.)
    - TRUE -> Nel caso in cui s'intenda generare un PAdES-T
    - FALSE -> In caso contrario
  - **signingTime** = Stringa utilizzata per la valorizzazione dell'entry /**M** dell'oggetto **Signature Dictionary** della busta PAdES. (NOTA: ARSS inserirà la data di sistema del Server su cui è in esecuzione.)
    - dd/MM/yyyy HH:mm:ss -> Nel caso in cui s'intenda specificare esplicitamente la data di firma
    - NULL -> Nel caso in cui s'intenda demandare ad ARSS la valorizzazione dell'attributo.
  - **Session_id** = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla. Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
  - **Identity** = Oggetto Auth contenente i dati per l'autenticazione del firmatario
  - **TypeHSM** = Stringa contenente il tipo di HSM. Va sempre valorizzata con la stringa 'COSIGN'
  - **TypeOtpAuth** = Stringa utilizzata per indicare il dominio di autenticazione. Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX). Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
    - firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
  - **User** = Stringa contenente lo username dell'utente di firma.
  - **UserPWD** = Stringa contenente la password dell'utente di firma.
  - **OtpPWD** = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
  - **ext_authtype =** Enum che consente di utilizzare il metodo di autenticazione esteso.
    - CNS2 -> Indica l'autenticazione tramite carta CNS
    - ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
    - ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
  - **ext_auth_blobvalue** = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
  - **ext_auth_value** = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
  - **delegated_user** = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe. In questo caso non deve essere specificata la password del Titolare della chiave di Firma
  - **delegated_password** = Stringa che contiene la password dell'utente delegato
  - **delegated_domain** = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
  - **tsa_identity** = Oggetto TSAAuth opzionale, contenente i dati per l'autenticazione Al Server TSS. Se non specificato il server userà l'account di default se configurato.
  - **user** = Stringa contenente la user dell'utenza di marcatura temporale
  - **password** = Stringa contenente la password dell'utenza di marcatura temporale
  - **Notifymail** = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di firma.
  - **Notify_id** = Stringa contenente l'id che l'utente intende associare alla sessione di firma per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME. Nel caso di firma di singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
  - **CertID** = RFU. Valorizzare con AS0
  - **profile** = RFU. Impostare a NULL
- **apparence** – Oggetto PdfSignApparence contenente gli ulteriori parametri di generazione della firma PAdES:
  - **Location** = Stringa utilizzata per la valorizzazione dell'entry /**Location** dell'oggetto **Signature Dictionary.** Ad esempio 'Roma'
  - **Reason** = Stringa utilizzata per la valorizzazione dell'entry /**Reason** dell'oggetto **Signature Dictionary.** Ad esempio 'Per approvazione'
  - **Page** = Intero che indica il numero di pagina nella quale deve essere inserita la firma. Il campo è obbligatorio
  - **Leftx** = Intero che serve a specificare l'ascissa del margine in basso a sinistra del rettangolo di firma. L'unità di misura è espressa in pixel
  - **Lefty** = Intero che serve a specificare l'ordinata del margine in basso a sinistra del rettangolo di firma. L'unità di misura è espressa in pixel
  - **Rightx** = Intero che serve a specificare l'ascissa del margine in alto a destra del rettangolo di firma. L'unità di misura è espressa in pixel
  - **Righty** = Intero che serve a specificare l'ordinata del margine in alto a destra del rettangolo di firma. L'unità di misura è espressa in pixel
  - **Image** = Stringa che indica il percorso dell'immagine da utilizzare come background nel rettangolo di firma. Il percorso indicato deve puntare ad una risorsa grafica in formato PNG/GIF/JPG raggiungibile dal server ARSS
  - **ImageBin** = Specifica il byte array dell'immagine da utilizzare come background nel rettangolo di firma. Se specificato ha precedenza rispetto al parametro Image. Disponibile dalla versione 1.14.0.
  - **Testo** = Stringa che permette di specificare testo libero al posto del testo standard riportato nella parte visibile delle firme.
  - **preservePDFA** = Booleano, se vero le immagini contenute nei campi Image o ImageBin vengono convertite eliminando eventuali trasparenze, in modo da preservare il formato PDF/A.
  - **ImageOnly** = Booleano, se vero inserisce solo l'immagine.
  - **bScaleFont** = Booleano, scala la dimensione del font del testo per consentire il fit all'interno della larghezza della box dell'apparence.
  - **bShowDateTime** = Booleano, mostra nasconde la data e l'ora di firma (signing time) nell'apparence.
  - **resizeMode** = Intero che specifica la modalità con la quale viene eseguito il resize dell'immagine. 1 (ridimensiona usando entrambi i lati dell'immagine – stretch), 2 (ridimensiona usando il lato maggiore e centra), 3 (ridimensiona usando il lato maggiore e allinea a destra), 4 (ridimensiona usando il lato maggiore e allinea a sinistra), 5 non ridimensionare
- **Pdfprofile** = Enum dei possibili tipi di PDF supportato:
  - BASIC -> Firma PDF Basic
  - PADESBES -> Firma PDF Pades-Bes
  - PADESLTV-> Firma PDF Pades-LT
- **password** = password di owner da specificare nel caso in cui il documento sia protetto da password.
- **dict_signed_attributes** - Oggetto DictionarySignedAttributes che permette di specificare propietà firmate nel dictionary di firma PDF. Disponibile dalla versione 1.14.0
  - **T** = Indiva il valore dell'attributo /T nel dictionary di firma (firma nominale).

- **signatureLevel** – Livello di firma richiesto. (Valore Ammesso: LT)

OUTPUT
Oggetto SignReturnV2 contenente i seguenti parametri:

- **Status** = Stringa contenente l'esito dell'operazione di firma
  - OK -> Operazione effettuata correttamente
  - Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignReturnV2.description e SignReturnV2.return_code.
- **Binaryoutput** = Byte array contenente il file firmato
- **Stream** = DataHandler per scaricare file di grandi dimensioni in formato Stream. Utilizzato esclusivamente nel caso di Transport = STREAM. (NOTA: La soluzione e disponibile solo con client JAX-WS compliant.)
- **DstPath** = Stringa contenente il percorso specificato all'interno del parametro SignRequestV2.DstName in fase di richiesta. Nel caso in cui sia il file specificato in SignRequestV2.DstName già esista la property SignReturnV2.DstPath contiene il percorso al file effettivamente generato da ARSS.
- **Return_code** = Enum contenente il codice di errore
  - "0001" -> Errore generico nel processo di firma
  - "0002" -> Parametri non corretti per il tipo di trasporto indicato
  - "0003" -> Errore in fase di verifica delle credenziali
  - "0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
  - "0005" -> Tipo di trasporto non valido
  - "0006" -> Tipo di trasporto non autorizzato
  - "0007" -> Profilo Di firma PDF non valido
  - "0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
  - "0009" -> Credenziali di delega non valide
  - "0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
- **Description** = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code

Firma Multipla

Per effettuare la firma digitale di un insieme di file in formato pdf valgono le stesse considerazioni già riportate nel paragrafo *Firma Multipla* d el capitolo **Firma Digitale remota in pkcs1**
Firma Digitale di un'impronta secondo standard pkcs1
Se s'intende effettuare la firma dell'impronta di un documento (o di una sequenza generica di byte) basata sullo statndard pkcs1 è sufficiente utilizzare il metodo signhash().
Tale metodo è così definito:

```
public SignHashReturn signhash(SignHashRequest request)
```

Questo metodo viene normalmente utilizzato per i seguenti scopi:

- Generazione di firme digitali "detached" cioè separate dal file firmato

INPUT

- **request** - Oggetto SignHashRequest contenente i seguenti parametri per la richiesta di firma:
  - **certID** = Identificativo univoco del certificato da utilizzare per la firma. L'elenco degli ID è recuperabile dalla funzione listCert esposta in seguito. In caso non si necessiti di un certificato specifico si può specificare AS0 al fine di ottenere il bilanciamento delle richieste, tale metodo però presume che il certificato stesso non concorra al calcolo dell'HASH (non valido per firma CADES).
  - **hash** = Array di byte contenente l'impronta del documento da firmare.
  - **hashtype** = Stringa contenente il nome dell'algoritmo di hash usato per il calcolo dell'impronta (attualmente disponibile solo "SHA256").
  - **Identity** = Oggetto Auth contenente i dati per l'autenticazione del firmatario
    - **TypeHSM** = Stringa contenente il tipo di HSM. Va sempre valorizzata con la stringa 'COSIGN'
    - **TypeOtpAuth** = Stringa utilizzata per indicare il dominio di autenticazione. Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
      - firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
    - **User** = Stringa contenente lo username dell'utente di firma.
    - **UserPWD** = Stringa contenente la password dell'utente di firma.
    - **OtpPWD** = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
    - **ext_authtype =** Enum che consente di utilizzare il metodo di autenticazione esteso.
      - CNS2 -> Indica l'autenticazione tramite carta CNS
      - ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
      - ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
    - **ext_auth_blobvalue** = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
    - **ext_auth_value** = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
    - **delegated_user** = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe. In

questo caso non deve essere specificata la password del Titolare della chiave di Firma
- **delegated_password** = Stringa che contiene la password dell'utente delegato
- **delegated_domain** = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)

- **session_id** = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla. Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
- **requirecert =** booleano che indica la richiesta della presenza del certificato del firmatario nella risposta.

OUTPUT
Oggetto SignHashReturn contenente i seguenti parametri:

- **Status** = Stringa contenente l'esito dell'operazione di firma
  - OK -> Operazione effettuata correttamente
  - Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignHashReturn.Description.
- **Description** = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code.
- **Return**_code = Enum contenente il codice di errore
  - "0001" -> Errore generico nel processo di firma
  - "0002" -> Parametri non corretti per il tipo di trasporto indicato
  - "0003" -> Errore in fase di verifica delle credenziali
  - "0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
  - "0005" -> Tipo di trasporto non valido
  - **"0006"** -> Tipo di trasporto non autorizzato
  - "0007" -> Profilo Di firma PDF non valido
  - "0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
  - "0009" -> Credenziali di delega non valide
  - "0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
- **signature** = Array di byte contenente il risultato del processo di firma.
- **cert** = Array di byte contenente certificato del destinatario se richiesto vedi parametro requirecert.
- **certID** = Identificativo del certificato del firmatario.

Firma Digitale remota in standard CAdES (p7m)
Le uniche differenze rispetto a quanto descritto per la firma CAdES sono le seguenti:

1. Il metodo da utilizzare per l'invocazione della funzione di firma dovrà essere pdfsignatureV2()e non pkcs7signV2()
2. Nel caso di tipo trasporto uguale a DIRECTORYNAME verranno processati solo i file con caratteristiche congruenti con i parametri specificati all'interno dell'oggetto apparence.
3. Quei file che non verranno processati saranno riepilogati all'interno della mail di notifica che verrà inviata al termine del processo alla casella specificata all'interno di Notitymail

Firma Automatica

Per effettuare una firma automatica in formato pdf è sufficiente utilizzare il metodo pdfsignatureV2() passando in input le credenziali di autenticazione di un'utenza di firma automatica.
INPUT
I dettagli di invocazione del metodo pdfsignatureV2()seguono le indicazioni già riportate nella relativa sezione del paragrafo *Firma Digitale* co n le seguenti modifiche contestualizzate per il processo di firma automatica:

1. Impostare la property SignRequestV2.Identity.TypeOtpAuth con la stringa identificativa del dominio di firma automatica.

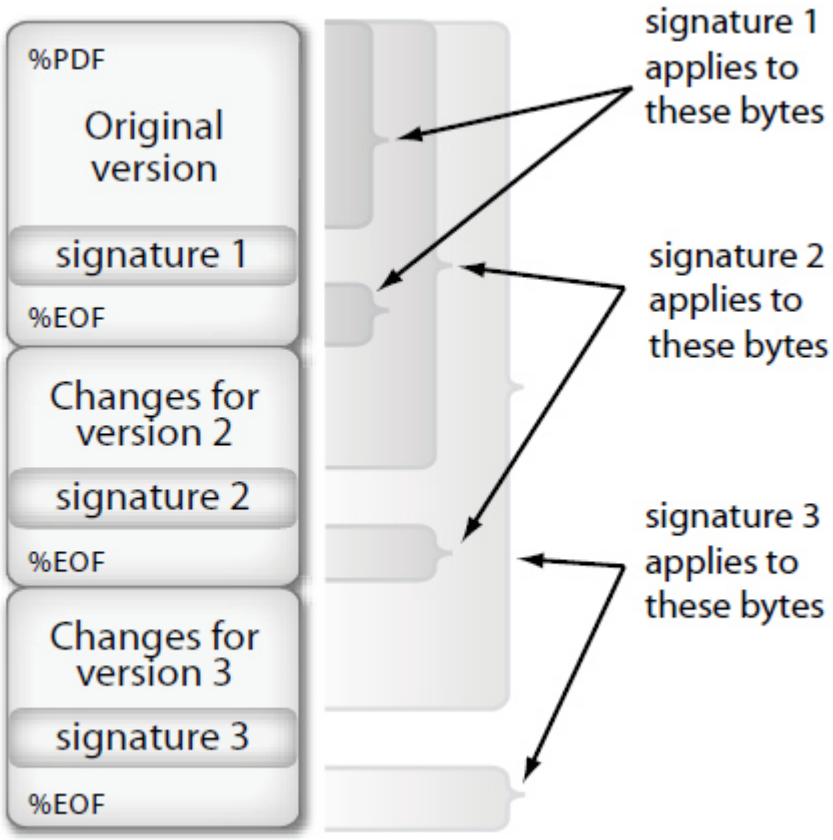
NOTA: Questa stringa viene indicata in fase di installazione dell'ARSS

1. Impostare la property SignRequestV2.Identity.OtpPWD con la stringa identificativa delle transazioni di firma automatica

NOTA: Questa stringa statica viene definita in fase di installazione del server ARSS ed è normalmente nota all'amministratore di rete dell'infrastruttura IT su cui opera l'utente.
OUTPUT
L'output restituito dalla chiamata a funzione segue quanto già indicato nella corrispondente sezione del paragrafo *Firma Digitale*

Firma Congiunta (firma parallela)

Lo standard PAdES non prevede l'utilizzo delle cosiddette firme congiunte, ovvero quelle firme apposte da diversi sottoscrittori allo stesso contenuto.
Nel formato PDF Ogni firma è infatti inserita in append utilizzando il cosiddetto sistema di "Incremental Update" tipico del formato PDF e, pertanto, copre l'intero contenuto del file PDF di input, comprese le eventuali firme già esistenti.
Di seguito è riportata una figura che illustra questo meccanismo.



L'unica operazione di firma che può quindi essere eseguita su un file PAdES è l'aggiunta di una firma in modalità "Enveloped" attraverso il semplice uso del metodo pdfsignatureV2()così come descritto al paragrafo *Firma Digitale* del Capitolo **Firma Digitale remota in standard PAdES (pdf).**
L'unica differenza è costituita dal fatto che in input deve essere passato un file in formato PAdES.

Firma con campo di firma compilabile

Per apporre più firme pdf sullo stesso documento ed utilizzare campi di firma precompilati è disponibile la seguente chiamata nel servizio SignService.

```
public SignResponse pdfSignature(PdfSignRequest
request)
```

INPUT

- **Binaryinput** = Byte array contenente il documento da firmare. Utilizzato esclusivamente nel caso di Transport = BINARYNET. NOT A = Il file di input deve essere un file xml well-formed
- **DstName** = Stringa contenente il path del file o della cartella di destinazione dei file firmati. Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
- **Notify_id** = Stringa contenente l'id che l'utente intende associare alla sessione di firma per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
- **SrcName** = Stringa contenenti il path del file o della cartella contenente i files da firmare. Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
- **Stream** = DataHandler per scaricare file di grandi dimensioni in formato Stream
- **Transport** = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
    - BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da firmare
    - FILENAME -> Nel caso in cui si stia passando in input il percorso del file da firmare
    - DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da firmare
    - STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)
- **Identity** = Oggetto Auth contenente i dati per l'autenticazione del firmatario
- **delegated_domain** = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
- **delegated_password** = Stringa che contiene la password dell'utente delegato

- **delegated_user** = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe. In questo caso non deve essere specificata la password del Titolare della chiave di Firma
- **ext_auth_blobvalue** = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
- **ext_auth_value** = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei

  metodi autenticazione estesa che prevedono lo scambio di dati in plain-text

- **ext_authtype =** Enum che consente di utilizzare il metodo di autenticazione esteso.
  - CNS2 -> Indica l'autenticazione tramite carta CNS
  - ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
  - ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms

- **OtpPWD** = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
- **TypeHSM** = Stringa contenente il tipo di HSM. Va sempre valorizzata con la stringa 'COSIGN'
- **TypeOtpAuth** = Stringa utilizzata per indicare il dominio di autenticazione firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota. Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX). Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
- **user** = Stringa contenente la user dell'utenza di marcatura temporale
- **UserPWD** = Stringa contenente la password dell'utente di firma. Nel caso di firma di singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
- **CertID** = RFU. Valorizzare con AS0
- **requiredmark** = Boolean utilizzato per pilotare la generazione di una busta PAdES-T: firma digitale con apposizione contestuale della marca temporale.
  - TRUE -> Nel caso in cui s'intenda generare un PAdES-T
  - FALSE -> In caso contrario

  NOTA: Nel caso in cui s'intenda effettuare la marca temporale in formato PAdES-T, il server ARSS dovrà essere preventivamente configurato con un account di marcatura temporale valido.

- **signingTime** = Stringa utilizzata per la valorizzazione dell'entry /**M** dell'oggetto **Signature Dictionary** della busta PAdES.
  - dd/MM/yyyy HH:mm:ss -> Nel caso in cui s'intenda specificare esplicitamente la data di firma
  - NULL -> Nel caso in cui s'intenda demandare ad ARSS la valorizzazione dell'attributo.

  Utilizzato esclusivamente nel caso di Transport = STREAM

  NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

- **sessionId** = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla. Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
- **tsaIdentity** = Oggetto TSAAuth opzionale, contenente i dati per l'autenticazione Al Server TSS. Se non specificato il server userà l'account di default se configurato.
  - **user** = Stringa contenente la user dell'utenza di marcatura temporale
  - **password =** Stringa contenente la password dell'utenza di marcatura temporale
  - **tsaurl =** Contiene l'indirizzo della TSA

- **apparence** – Oggetto PdfSignApparence contenente gli ulteriori parametri di generazione della firma PadES
- **Image** = Stringa che indica il percorso dell'immagine da utilizzare come background nel rettangolo di firma. Il percorso indicato deve puntare ad una risorsa grafica in formato PNG/GIF/JPG raggiungibile dal server ARSS
- **ImageBin** = Specifica il byte array dell'immagine da utilizzare come background nel rettangolo di firma. Se specificato ha precedenza rispetto al parametro Image. Disponibile dalla versione 1.14.0.
- **ImageOnly** = Booleano, se vero inserisce solo l'immagine.
- **Leftx** = Intero che serve a specificare l'ascissa del margine in basso a sinistra del rettangolo di firma. L'unità di misura è espressa in pixel
- **Lefty** = Intero che serve a specificare l'ordinata del margine in basso a sinistra del rettangolo di firma. L'unità di misura è espressa in pixel
- **Location** = Stringa utilizzata per la valorizzazione dell'entry /Location dell'oggetto Signature Dictionary. Ad esempio 'Roma'
- **Page** = Intero che indica il numero di pagina nella quale deve essere inserita la firma. Il campo è obbligatorio
- **Reason** = Stringa utilizzata per la valorizzazione dell'entry /Reason dell'oggetto Signature Dictionary. Ad esempio 'Per approvazione'
- **Rightx** = Intero che serve a specificare l'ascissa del margine in alto a destra del rettangolo di firma. L'unità di misura è espressa in pixel
- **Righty** = Intero che serve a specificare l'ordinata del margine in alto a destra del rettangolo di firma. L'unità di misura è espressa in pixel
- **Testo** = Stringa che permette di specificare testo libero al posto del testo standard riportato nella parte visibile delle firme.
- **bScaleFont** = Booleano, scala la dimensione del font del testo per consentire il fit all'interno della larghezza della box dell'apparence.
- **bShowDateTime** = Booleano, mostra nasconde la data e l'ora di firma (signing time) nell'apparence.
- **resizeMode** = Intero che specifica la modalità con la quale viene eseguito il resize dell'immagine. 1 (ridimensiona usando entrambi i lati dell'immagine – stretch), 2 (ridimensiona usando il lato maggiore e centra), 3 (ridimensiona usando il lato maggiore e allinea a destra), 4 (ridimensiona usando il lato maggiore e allinea a sinistra), 5 non ridimensionare
- **preservePDFA** = Booleano, consente di preservare il PDF/A
- **fieldName =** Stringa che indica il nome del campo di firma precompilato sul quale creare l'appareance PDF.
- **Pdfprofile** = Enum dei possibili tipi di PDF supportato:

        - BASIC -> Firma PDF Basic
        - PADESBES -> Firma PDF Pades-Bes
        - PADESLTV-> Firma PDF Pades-LT
- **dictSignedAttributes** - Oggetto DictionarySignedAttributes che permette di specificare propietà firmate nel dictionary di firma PDF. Disponibile dalla versione 1.14.0
- **T** = Indiva il valore dell'attributo /T nel dictionary di firma (firma nominale).

Firma Digitale remota in standard XAdES (xml)

Brevi richiami sulla firma XML

Lo XML ha una vastissima diffusione come formato per lo scambio di dati in modo efficace e flessibile tra applicazioni che interagiscono in rete.
Lo standard "XML-Signature Syntax and Processing" (http://www.w3.org/TR/xmldsig-core/), definisce le specifiche che permettono di "firmare" documenti XML o parti di essi, garantendone l'autenticità e l'integrità. In questo modo le applicazioni possono realizzare funzionalità di generazione e verifica di firma digitale analogamente a quanto già avviene per le buste CAdES e PAdES. È importante sottolineare che un documento XML firmato è ancora un documento XML.
Il processo di firma XML prevede cinque passi:

- identificare il documento da firmare;
- trasformarlo in forma "canonica";
- applicare le eventauli trasformazioni (XPath, XSLT);
- creare un sommario (digest) dei risultati della trasformazione;
- firmare il digest.

XML-Signature, è la struttura prevista nello standard XML per supportare la firma digitale.
Questa struttura, che accoglie la firma digitale in un file XML, è chiamata **<Signature>** ed è costituita da tre elementi principali:
<SignedInfo>
<SignatureValue>
<KeyInfo>
Il primo elemento, <SignedInfo>, racchiude le informazioni su come è stata generata la firma:

- l'algoritmo utilizzato per ottenere la forma canonica XML
- l'algoritmo utilizzato per produrre la firma
- le informazioni sulla risorsa firmata.

Queste ultime informazioni sulla risorsa firmata sono specificate all'interno dell'elemento **<Reference>** e riguardano l'URI del documento XML, l'algoritmo utilizzato per calcolarne il digest e il valore del digest stesso.
L'elemento **<SignatureValue>** contiene il valore della firma digitale ottenuta secondo l'algoritmo indicato nell'elemento precedente.
L'ultimo elemento, **<KeyInfo>**, contiene le informazioni sulla chiave pubblica del firmatario e viene utilizzata per decifrare la firma digitale XML nella fase di verifica dell'autenticità e dell'integrità del documento.
È importante ricordare che lo standard prevede differenti tipi di Firma XML:

- Enveloping (http://www.w3.org/TR/xmldsig-core/#def-SignatureEnveloping)
- Enveloped (http://www.w3.org/TR/xmldsig-core/#def-SignatureEnveloped)
- Detached (http://www.w3.org/TR/xmldsig-core/#def-SignatureDetached)

Si parla di **Enveloping Signature** quando la firma, cioè l'elemento **<Signature>,** contiene l'oggetto firmato.
Si parla invece di **Enveloped Signature** quando l'oggetto firmato contiene l'elemento **<Signature>**.
Infine abbiamo una **Detached Signature** quando l'elemento **<Signature>** non è ne un "parent" ne un "descendant" dell'oggetto firmato.
Quest'ultima casistica si può ulteriormente declinare in:

- **Detached Internal Signature** nel caso in cui l'elemento **<Signature>** sia contenuta all'interno dello stesso documento xml che contiene l'oggetto firmato e i due (firma e oggetto firmato) si trovano in un rapporto cosiddetto di "sibling"
- **Detached External Signature** nel caso in cui l'elemento <Signature> sia contenuta all'interno di un documento xml diverso da quello che contiene l'oggetto firmato.

Firma Digitale

Il metodo da utilizzare per generare una firma digitale di un singolo file in formato XAdES è xmlsignature()**.**
Il metodo è così definito:

```
public SignReturnV2 xmlsignature(SignRequestV2 request, XmlSignatureParameter parameter)
```

INPUT

request - Oggetto SignRequestV2 contenente i seguenti parametri per la richiesta di firma
Transport = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da firmare
FILENAME -> Nel caso in cui si stia passando in input il percorso del file da firmare
DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da firmare
STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)

Binaryinput = Byte array contenente il documento da firmare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
NOTA = Il file di input deve essere un file xml well-formed
SrcName = Stringa contenenti il path del file o della cartella contenente i files da firmare.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA = Il file di input deve essere un file xml well-formed
NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS
Stream = DataHandler per caricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

DstName = Stringa contenente il path del file o della cartella di destinazione dei file firmati.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS
requiredmark = RFU, non utilizzare
signingTime = Stringa utilizzata per la valorizzazione dell'elemento <xsd:SigningTime> da inserire nella busta XADES.
Al momento l'unica valorizzazione possibile per il parametro è NULL per indicare al server ARSS che l'elemento deve essere popolato tramite l'inserimento della data di sistema del Server su cui è in esecuzione.
Session_id = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla.
Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
Identity = Oggetto Auth contenente i dati per l'autenticazione del firmatario
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
ext_authtype = Enum che consente di utilizzare il metodo di autenticazione esteso.
CNS2 -> Indica l'autenticazione tramite carta CNS
ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
ext_auth_blobvalue = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
ext_auth_value = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
delegated_user = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe.
In questo caso non deve essere specificata la password del Titolare della chiave di Firma
delegated_password = Stringa che contiene la password dell'utente delegato
delegated_domain = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
tsa_identity = Oggetto TSAAuth opzionale, contenente i dati per l'autenticazione Al Server TSS. Se non specificato il server userà l'account di default se configurato.
user = Stringa contenente la user dell'utenza di marcatura temporale
password = Stringa contenente la password dell'utenza di marcatura temporale
Notitymail = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di firma.
Notity_id = Stringa contenente l'id che l'utente intende associare alla sessione di firma per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
Nel caso di firma di singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
CertID = RFU. Valorizzare con AS0
profile = RFU. Impostare a NULL

parameter – Oggetto di tipo XmlSignatureParameter con i seguenti parametri:
Type = Enum utilizzata per indicare il tipo di firma XML da effettuare:
XMLENVELOPED -> Nel caso in cui si intenda effettuare una firma di tipo **Enveloped**
XMLENVELOPING -> Nel caso in cui si intenda effettuare una firma di tipo **Enveloping**
XMLDETACHED_INTERNAL -> Nel caso in cui si intenda effettuare una firma di tipo **Detached Internal**
Transforms = Array di oggetti Transform che indica la lista ordinata delle trasformazioni da applicare al documento prima del calcolo dell'hash.
Ogni elemento Transform è composto dai seguenti campi:
Type = Enum che indica il tipo di trasformazione
CANONICAL_WITH_COMMENT,
CANONICAL_OMIT_COMMENT,
BASE64,
XPATH2_INTERSECT,
XPATH2_SUBTRACT,
XPATH2_UNION,
XSLT
Value = Eventuale valore da inserire a seconda del tipo di trasformazione specificata in Type

Ad esempio espressioni Xpath etc…
canonicalizedType = Enum che indica il tipo di canonicalizzazione.
Possibili valori:
ALGO_ID_C14N11_OMIT_COMMENTS,
ALGO_ID_C14N11_WITH_COMMENTS

signatureLevel – Livello di firma richiesto. (Valore Ammesso: LT)

OUTPUT
Oggetto SignReturnV2 contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di firma
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignReturnV2.description e SignReturnV2.return_code.
Binaryoutput = Byte array contenente il file firmato
Stream = DataHandler per scaricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

DstPath = Stringa contenente il percorso specificato all'interno del parametro SignRequestV2.DstName in fase di richiesta.
Nel caso in cui sia il file specificato in SignRequestV2.DstName già esista la property SignReturnV2.DstPath contiene il percorso al file effettivamente generato da ARSS.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di firma
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)

Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code

Firma Multipla

Per effettuare la firma digitale di un insieme di file in formato xml valgono le stesse considerazioni già riportate nel paragrafo *Firma Multipla* d el capitolo **Firma Digitale remota in pkcs1**
Firma Digitale di un'impronta secondo standard pkcs1
Se s'intende effettuare la firma dell'impronta di un documento (o di una sequenza generica di byte) basata sullo statndard pkcs1 è sufficiente utilizzare il metodo signhash().
Tale metodo è così definito:

```
public SignHashReturn signhash(SignHashRequest request)
```

Questo metodo viene normalmente utilizzato per i seguenti scopi:

- Generazione di firme digitali "detached" cioè separate dal file firmato

INPUT
request - Oggetto SignHashRequest contenente i seguenti parametri per la richiesta di firma
Identity = Oggetto Auth contenente i dati per l'autenticazione del firmatario
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
ext_authtype = Enum che consente di utilizzare il metodo di autenticazione esteso.
CNS2 -> Indica l'autenticazione tramite carta CNS
ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
ext_auth_blobvalue = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
ext_auth_value = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che

prevedono lo scambio di dati in plain-text
delegated_user = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe.
In questo caso non deve essere specificata la password del Titolare della chiave di Firma
delegated_password = Stringa che contiene la password dell'utente delegato
delegated_domain = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)

certID = Identificativo univoco del certificato da utilizzare per la firma. L'elenco degli ID è recuperabile dalla funzione listCert esposta in seguito.
In caso non si necessiti di un certificato specifico si può specificare AS0 al fine di ottenere il bilanciamento delle richieste, tale metodo però presume che il certificato stesso non concorra al calcolo dell'HASH (non valido per firma CADES).
hash = Array di byte contenente l'impronta del documento da firmare.
hashtype = Stringa contenente il nome dell'algoritmo di hash usato per il calcolo dell'impronta (attualmente disponibile solo "SHA256").
session_id = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla.
Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
requirecert = booleano che indica la richiesta della presenza del certificato del firmatario nella risposta.
OUTPUT
Oggetto SignHashReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di firma
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignHashReturn.Description.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di firma
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0007" -> Profilo Di firma PDF non valido
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
signature = Array di byte contenente il risultato del processo di firma.
cert = Array di byte contenente certificato del destinatario se richiesto vedi parametro requirecert.
certID = Identificativo del certificato del firmatario.
Firma Digitale remota in standard CAdES (p7m)
Le uniche differenze rispetto a quanto descritto per la firma CAdES sono le seguenti:

1. Il metodo da utilizzare per l'invocazione della funzione di firma dovrà essere xmlsignature()e non pkcs7signV2()
2. Nel caso di tipo trasporto uguale a DIRECTORYNAME verranno processati solo i file xml con caratteristiche congruenti con i parametri specificati all'interno dell'oggetto parameter.
3. Quei file che non verranno processati saranno riepilogati all'interno della mail di notifica che verrà inviata al termine del processo alla casella specificata all'interno di Notitymail

Firma Automatica

Per effettuare una firma automatica in formato xml è sufficiente utilizzare il metodo xmlsignature()passando in input le credenziali di autenticazione di un'utenza di firma automatica.
INPUT
I dettagli di invocazione del metodo xmlsignature()seguono le indicazioni già riportate nella relativa sezione del paragrafo *Firma Digitale* con le seguenti modifiche contestualizzate per il processo di firma automatica:

1. Impostare la property SignRequestV2.Identity.TypeOtpAuth con la stringa identificativa del dominio di firma automatica.

NOTA: Questa stringa viene indicata in fase di installazione dell'ARSS

1. Impostare la property SignRequestV2.Identity.OtpPWD con la stringa identificativa delle transazioni di firma automatica

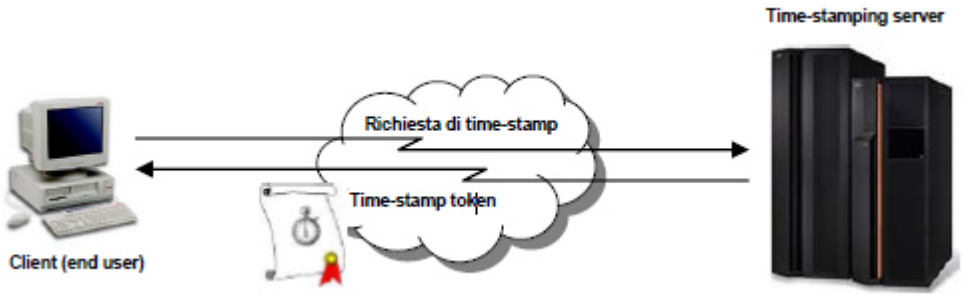
NOTA: Questa stringa statica viene definita in fase di installazione del server ARSS ed è normalmente nota all'amministratore di rete dell'infrastruttura IT su cui opera l'utente.
OUTPUT
L'output restituito dalla chiamata a funzione segue quanto già indicato nella corrispondente sezione del paragrafo *Firma Digitale* del presente Capitolo

Marcatura temporale (time stamping)

Il server ARSS permette di accedere a servizi di digital time stamping (o "marcatura temporale") conformi allo standard rfc 3161. In particolare, con ARSS è possibile richiedere il time stamping per dati qualsiasi.

Brevi richiami sul time stamping

Un servizio di digital time stamping (DTS) gioca un ruolo importante ai fini del non-ripudio delle transazioni. In particolare, lo scopo di un servizio DTS è quello di dimostrare che un certo dato D di interesse (messaggio, buffer, documento, etc) esisteva non più tardi di un certo istante T di tempo.
Questo meccanismo può essere usato, per esempio, per verificare che una firma digitale è stata generata prima che il corrispondente certificato venisse revocato; in tal caso, il certificato revocato può comunque essere utilizzato per verificare firme digitali create prima della data/ora di revoca. Si tratta di un importante operazione nell'ambito delle PKI (public key infrastructure).
Un altro importante utilizzo della marcatura temporale si ha nelle applicazioni di "deposito" di documenti (es. bilanci, brevetti, dichiarazioni fiscali, etc) per via telematica, quando il momento dell'invio o della ricezione sono di importanza critica.
Il meccanismo alla base del digital time stamping è molto semplice. Essenzialmente, la parte richiedente (client) calcola l'hash H del dato originale D, quindi invia il valore H ad un server di time stamping (gestito da una terza parte fidata che eroga il servizio, chiamata Time Stamping Authority o TSA). Il server concatena il valore H con il tempo corrente T (data e ora) ottenuto da una fonte affidabile, quindi firma il tutto con la propria chiave privata. Il risultato, chiamato time-stamp token, viene restituito al client. La seguente figura illustra la semplice interazione su cui si basa un servizio di time-stamping:



Lo standard Rfc 3161 definisce il protocollo di comunicazione fra client e server, ovvero le strutture dati scambiate e le modalità di trasporto.
Il supporto al time-stamping offerto da ARSS si basa appunto su tale standard e consente una facile integrazione "mascherando" tramite singole chiamate SOAP tutti i dettagli di basso livello del protocollo Rfc 3161 per la richiesta di marche temporali
Per poter effettuare con successo marche temporali tramite ARSS è necessario aver preventivamente attivato un'utenza di marcatura temporale e avere a disposizione le relative credenziali d'accesso.

Marca temporale in formato TSR (.tsr)

Il metodo da utilizzare per generare una marca temporale in formato tsr scollegata dal file marcato è tsr().
Il metodo è così definito:

```
public MarkReturn tsr (MarkRequest request)
```

INPUT
request - Oggetto MarkRequest contenente i seguenti parametri per la richiesta di timestamp
Transport = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da marcare
FILENAME -> Nel caso in cui si stia passando in input il percorso del file da marcare
DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da marcare
STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)

Binaryinput = Byte array contenente il documento da marcare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
SrcName = Stringa contenenti il path del file o della cartella con i files da marcare.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS
Stream = DataHandler per caricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.
DstName = Stringa contenente il path del file o della cartella di destinazione dei file marcati.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS
Notitymail = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di marcatura temporale.
Notity_id = Stringa contenente l'id che l'utente intende associare alla sessione di marcatura temporale per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
Nel caso di marcatura di un singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
User = Stringa contenente la user dell'utenza di marcatura temporale
Password = Stringa contenente la password dell'utenza di marcatura temporale
OUTPUT
Oggetto MarkReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente

Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property MarkReturn.description e MarkReturn.return_code.

Binaryoutput = Byte array contenente la marca temporale in formato tsr
Stream = DataHandler per scaricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

DstPath = Stringa contenente il percorso specificato all'interno del parametro MarkRequest.DstName in fase di richiesta.
Nel caso in cui il file specificato in MarkRequest.DstName già esista, la property MarkReturn.DstPath punterà al file effettivamente generato da ARSS.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di timestamp
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato
"0007" -> Profilo Di firma PDF non valido
"0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
"0009" -> Credenziali di delega non valide
"0010" -> Lo stato dell'utente non è valido (es. utente sospeso)

Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property MarkReturn.return_code

Marca temporale in formato TSD (.tsd)

Il metodo da utilizzare per generare una marca temporale unita al file marcato in formato tsd è tsd().
Con questo metodo ARSS permette la completa gestione di "buste marcate temporalmente" nel rispetto della specifica pubblica RFC 5544 ("TimeStampedData").
Il formato TimeStampedData consente di "tenere assieme" un file qualsiasi (non necessariamente firmato) con una marca temporale. In questo modo si può evitare l'uso di formato proprietari e non interoperabili.
Il metodo è così definito:

```
public MarkReturn tsd (MarkRequest request)
```

INPUT
request - Oggetto MarkRequest contenente i seguenti parametri per la richiesta di timestamp
Transport = Enum che indica il tipo di trasporto
BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da marcare
FILENAME -> Nel caso in cui si stia passando in input il percorso del file da marcare
DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da marcare
Binaryinput = Byte array contenente il documento da marcare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
SrcName = Stringa contenenti il path del file o della cartella con i files da marcare.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS
DstName = Stringa contenente il path del file o della cartella di destinazione dei file marcati.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS
Stream = DataHandler per caricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

Notitymail = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di marcatura temporale.
Notity_id = Stringa contenente l'id che l'utente intende associare alla sessione di marcatura temporale per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
Nel caso di marcatura di un singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
User = Stringa contenente la user dell'utenza di marcatura temporale
Password = Stringa contenente la password dell'utenza di marcatura temporale
OUTPUT
Oggetto MarkReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property MarkReturn.description e MarkReturn.return_code.

Binaryoutput = Byte array contenente la marca temporale in formato tsd
Stream = DataHandler per scaricare file di grandi dimensioni in formato Stream
Utilizzato esclusivamente nel caso di Transport = STREAM
NOTA: La soluzione e disponibile solo con client JAX-WS compliant.

DstPath = Stringa contenente il percorso specificato all'interno del parametro MarkRequest.DstName in fase di richiesta.
Nel caso in cui il file specificato in MarkRequest.DstName già esista, la property MarkReturn.DstPath punterà al file effettivamente generato da ARSS.

Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di timestamp
"0002" -> Parametri non corretti per il tipo di trasporto indicato
"0003" -> Errore in fase di verifica delle credenziali
"0005" -> Tipo di trasporto non valido
"0006" -> Tipo di trasporto non autorizzato

Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property MarkReturn.return_code

Marca temporale in formato M7M (.m7m)

METODO RIMOSSO

Verifica della firma digitale

Dalla Versione ARSS 1.13.1 i metodi utilizzati per la verifica sono stati rimossi. Il prodotto di riferimento per le operazioni di verifica diventa VOL. Conttattare il proprio commerciale per maggiori dettagli.

Funzioni ausiliarie

Lista dei certificati associati ad un utente (via credenziali utente)

Il metodo da utilizzare per avere la lista dei certificati associati ad un utente attraverso le sue credenziale è listCert().
Il metodo è così definito.

```
public UserCertList listCert(Auth identity)
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = NULL
OUTPUT
Oggetto userCertList contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata
OK -> Ricerca effettuata con successo
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property userCertList.description.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
App1 = Oggetto di tipo List<Cert> contenente il base64 dei certificati configurati sul primo HSM
App2 = Oggetto di tipo List<Cert> contenente il base64 dei certificati configurati sul secondo HSM

Lista dei certificati associati ad un utente (via credenziali amministrative)

Il metodo da utilizzare per avere la lista dei certificati associati ad un utente attraverso credenziali amministrative è listCertAuth().
Il metodo è così definito.

```
public UserCertListAuth listCertAuth(ApplicationAuth identity, String domain, String user)
```

INPUT
identity - Oggetto ApplicationAuth contenente i dati per l'autenticazione dell'amministratore
applicationuser = Utente amministratore
applicationpassword = Password amministratore

domain - Dominio a cui appartiene l'utente di cui si vogliono cercare i certificati
user - Utente di cui cercare i certificati
OUTPUT
Oggetto UserCertListAuth contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata
OK -> Ricerca effettuata con successo
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property userCertList.description.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
certs = Oggetto di tipo List<Cert> contenente il base64 dei certificati configurati sul primo HSM

Lista dei metodi di autenticazione di un utente

Il metodo da utilizzare per avere la lista dei metodi di autenticazione di un utente è authMethods().
Il metodo è così definito.

```
public AuthMethodsReturn authMethods(Auth identity)
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stiano richiedendo le informazioni per un'utenza di firma remota.
Nel caso in cui si stiano invece richiedendo informazioni per un'utenza di firma automatica.
E' necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = NULL
OUTPUT
Oggetto AuthMethodsReturn contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata
OK -> Ricerca effettuata con successo
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property userCertList.description.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno, valorizzata a "0000" in caso di successo o ad un altro valore in caso di errore.
methods = Lista di stringhe che identificano i metodi di autenticazione dell'utente specificato

Lista dei processi di firma attivi

Nel caso in cui sia stata avviata la firma di cartelle il metodo listprocess()può essere utilizzato per avere un elenco delle operazioni ancora in corso.
Il metodo è così definito.

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="b7d12556-ef94-4994-ba54-b28a41ea7ac3"><ac:plain-text-body><![CDATA[public String[] listprocess(Auth identity)
```

```
]]></ac:plain-text-body></ac:structured-macro>
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = NULL
OUTPUT
Arrary di stringhe contenente l'elenco dei processi di firma in corso.
Ogni elemento dell'array contiene la stringa specificata all'interno del campo SignRequestV2.Session_id in fase di avvio del particolare processo di firma massiva di cartelle.
Quei processi ai quali non è stato preventimante associato alcun ID verranno mappati all'interno dell'array con valori randomici generati dal server ARSS.

Recupero versione ARSS

Il metodo da utilizzare per conoscere la versione del server ARSS in esercizio è getVersion()
Il metodo è così definito.

```
public String getVersion()
```

INPUT
Il metodo non ha parametri.
OUTPUT
Stringa descrittiva della versione di ARSS in uso.

Utility per la monitoria di ARSS

Il metodo da utilizzare per verificare il raggiungimento e l'operatività del server ARSS è ping()
Il metodo è così definito.

```
public String ping()
```

INPUT
Il metodo non ha parametri.
OUTPUT
Stringa 'pong' nel caso in cui il server ARSS sia raggiungibile ed operativo.

Utility per autenticazione con OTP delle utenze

Il metodo da utilizzare per autenticare un utenza di firma remota/automatica è verifyOtp()
Il metodo è così definito.

```
public ArssReturn verifyOtp(Auth identity)
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = OTP da verificare.
OUTPUT
Oggetto ArssReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property ArssReturn.description e ArssReturn.return_code.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di autenticazione via OTP.
Return_code = Enum contenente il codice di errore
"0001" -> Errore generico nel processo di autenticazione
"0002" -> Parametri non corretti
"0003" -> Errore in fase di verifica delle credenziali
"0004" -> Errore nel PIN

Utility per invio di una delle credenziali di firma (OTP) via SMS o identificativo chiamata (ARUBACALL)

Il metodo da utilizzare per l'invio dell'OTP all'utenza è sendCredential()
Il metodo è così definito.

```
public ArssReturn sendCredential(Auth identity, CredentialsType type)
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.

User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = NULL
type - Oggetto CredentialsType contenente i dati per l'identificazione del tipo autenticazione dell'utente. Si tratta a tutti gli effetti di un ENUM avente due possibili valori
"ARUBACALL"
"SMS"
"PAPERTOKEN" **Discontinuato, NON usare**
OUTPUT
Oggetto ArssReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property ArssReturn.description e ArssReturn.return_code.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di invio della credenziale OTP
Return_code = Enum contenente il codice di errore
"0002" -> Parametri non corretti
"0003" -> Errore in fase di verifica delle credenziali

Utility per il recupero di una credenziale (OTP PAPERTOKEN)

ATTENZIONE: Il metodo di autneticazione PAPERTOKEN è stato discontinuato! NON usare questo metodo!
Il metodo da utilizzare per il recupero di informazioni complementari dell'OTP è retriveCredential()
Il metodo è così definito.

```
public ArssReturn retriveCredential(Auth identity, CredentialsType type)
```

INPUT
identity - Oggetto Auth contenente i dati per l'autenticazione dell'utente
TypeHSM = Stringa contenente il tipo di HSM.
Va sempre valorizzata con la stringa 'COSIGN'
TypeOtpAuth = Stringa utilizzata per indicare il dominio di autenticazione
firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
User = Stringa contenente lo username dell'utente di firma.
UserPWD = Stringa contenente la password dell'utente di firma.
OtpPWD = NULL
type - Oggetto CredentialsType contenente i dati per l'identificazione del tipo autenticazione di cui l'utente desidera recuperare le informazioni complementari sull'OTP. Per ora supportato solo PAPERTOKEN.
OUTPUT
Oggetto RetriveCredentialReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property ArssReturn.description e ArssReturn.return_code.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di invio della credenziale OTP
Return_code = Enum contenente il codice di errore
"0002" -> Parametri non corretti
"0003" -> Errore in fase di verifica delle credenziali
blob = Se previsto dal metodo di autenticazione contiene il byte array necessario a recuperare la credenziale
textvalue = Se previsto dal metodo di autenticazione contiene il testo necessario a recuperare la credenziale. Ad Esempio per PAPERTOKEN contiene la lista delle coordinate.

Cifratura file

Il metodo di utilità per cifrare file.

```
public EncryptedEnvelopReturn encryptedEnvelope(EncryptedEnvelopReq request)
```

INPUT
request - Oggetto EncryptedEnvelopReq
user = utente del servizio di cifratura (deve essere configurato staticamente nelle opzioni di ARSS)
password = password del servizio di cifratura (deve essere configurata staticamente nelle opzioni di ARSS)
Transport = Enum che indica il tipo di trasporto
BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da marcare
FILENAME -> Nel caso in cui si stia passando in input il percorso del file da marcare
DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da marcare
Binaryinput = Byte array contenente il documento da cifrare. Utilizzato esclusivamente nel caso di Transport = BINARYNET
SrcName = Stringa contenenti il path del file o della cartella con i files da cifrare.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA = Il file di input deve essere un file firmato
NOTA: La risorsa indicata deve essere raggiungibile dal server ARSS
DstName = Stringa contenente il path del file o della cartella di destinazione dei file cifrati.
Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME
NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS
Notitymail = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di marcatura temporale.
Notity_id = Stringa contenente l'id che l'utente intende associare alla sessione di marcatura temporale per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME
Nel caso di marcatura di un singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
Recipients = Lista fi array di byte contenente i certificati da usare per la cifratura.
Algorithm = Enum che indica il tipo di algoritmo da usare per la cifratura
DES_EDE3_CBC Triple DES CBC
RC2_CBC RC2 CBC
IDEA_CBC IDEA CBC
CAST5_CBC CAST5 CBC
AES128_CBC AES 128 CBC
AES192_CBC AES 192 CBC
AES256_CBC AES 256 CBC
CAMELLIA128_CBC CAMELLIA 128 CBC
CAMELLIA192_CBC CAMELLIA 192 CBC
CAMELLIA256_CBC CAMELLIA 256 CBC
N.B. Non tutti gli algoritmo potrebbero essere abialitati a seconda della configurazione JDK.
OUTPUT
Oggetto EncryptedEnvelopReturn contenente i seguenti parametri:
Status = Stringa contenente l'esito dell'operazione di marcatura
OK -> Operazione effettuata correttamente
Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property ArssReturn.description e ArssReturn.return_code.
Description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di invio della credenziale OTP
Return_code = Enum contenente il codice di errore
"0002" -> Parametri non corretti
" 0011" -> Formato del certificato non valido
" 0012" -> Key usage del certificato non valido
" 0013" -> Algoritmo non valido.
binaryoutput = Se previsto dal metodo di trasporto contine il byte array cifrato in formato CMS.
dstPath = Se previsto dal metodo di trasporto contine il path del file cifrato.
stream = Se previsto dal metodo di trasporto contine lo stream contenten il cifrato in formato CMS.

Firma header soap WS-SECURITY

Il metodo da utilizzare è *soapsignature*.

```
public SignReturnV2 soapsignature(SignRequestV2 request)
```

INPUT

- **request** - Oggetto SignRequestV2 contenente i seguenti parametri per la richiesta di firma
  - **Transport** = Enum che indica la tipologia di input (da qui in avanti tipo trasporto)
    - BYNARYNET -> Nel caso in cui si stia passando in input l'array di byte contenente il file da firmare
    - FILENAME -> Nel caso in cui si stia passando in input il percorso del file da firmare
    - DIRECTORYNAME -> Nel caso in cui si stia passando in input il percorso della cartella con i files da firmare
    - STREAM -> Nel caso in cui si stia passando in input lo strema di un file in formato MTOM con tecnologia java JAX-WS (per file di grandi dimensioni)
  - **Binaryinput** = Byte array contenente il documento da firmare. Utilizzato esclusivamente nel caso di Transport = BINARYNET. (NOTA: Il file di input deve essere un file xml well-formed)
  - **SrcName** = Stringa contenenti il path del file o della cartella contenente i files da firmare. Utilizzato esclusivamente nel caso

di Transport = FILENAME o DIRECTORYNAME. (NOTA: Il file di input deve essere un file xml well-formed, NOTA: La risorsa indicata deve essereraggiungibile dal server ARSS)
- **Steam** = DataHandler per caricare file di grandi dimensioni in formato Stream. Utilizzato esclusivamente nel caso di Transport = STREAM. (NOTA: La soluzione e disponibile solo con client JAX-WS compliant.)
- **DstName** = Stringa contenente il path del file o della cartella di destinazione dei file firmati. Utilizzato esclusivamente nel caso di Transport = FILENAME o DIRECTORYNAME. (NOTA: Il percorso specificato deve essere raggiungibile dal server ARSS)
- **requiredMark** = RFU, non utilizzare
- **signingTime** = Stringa utilizzata per la valorizzazione dell'elemento <xsd:SigningTime> da inserire nella busta XADES. Al momento l'unica valorizzazione possibile per il parametro è NULL per indicare al server ARSS che l'elemento deve essere popolato tramite l'inserimento della data di sistema del Server su cui è in esecuzione.
- **Session_id** = Stringa utilizzata per specificare l'id di sessione all'interno di transazioni di firma multipla. Nel caso di firma di un singolo file o di una cartella non specificare ovvero valorizzare a NULL.
- **Identity** = Oggetto Auth contenente i dati per l'autenticazione del firmatario
- **TypeHSM** = Stringa contenente il tipo di HSM. Va sempre valorizzata con la stringa 'COSIGN'
- **TypeOtpAuth** = Stringa utilizzata per indicare il dominio di autenticazione
    - firma -> Nel caso in cui si stia utilizzando un'utenza di firma remota.
    - Nel caso in cui si stia utilizzando un'utenza di firma remota con dominio personalizzato è necessario specificare tale dominio (Es. frXXXX).
    - Nel caso in cui si stia utilizzando un'utenza di firma automatica è necessario specificare la stringa indicata in fase di installazione di ARSS.
- **User** = Stringa contenente lo username dell'utente di firma.
- **UserPWD** = Stringa contenente la password dell'utente di firma.
- **OtpPWD** = Stringa contenente il codice OTP dell'utente valido per la transazione di firma.
- **ext_authtype** = Enum che consente di utilizzare il metodo di autenticazione esteso.
    - CNS2 -> Indica l'autenticazione tramite carta CNS
    - ARUBACALL -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBACALL
    - ARUBASMS -> Indica l'autenticazione tramite OTP inviato tramite servizio ARUBsms
- **ext_auth_blobvalue** = Array di byte in cui passare la credenziale di autenticazione (challenge cifrata, etc..) per quei *metodi di autenticazione estesa* che prevedono lo scambio di dati binari
- **ext_auth_value** = Stringa in cui passare la credenziale di autenticazione (OTP o simile) per quei *metodi autenticazione estesa* che prevedono lo scambio di dati in plain-text
- **delegated_user** = Stringa che contiene la user dell'utente delegato nel caso di firma automatica tramite deleghe. In questo caso non deve essere specificata la password del Titolare della chiave di Firma
- **delegated_password** = Stringa che contiene la password dell'utente delegato
- **delegated_domain** = Dominio dell'utente delegato (utilizzo simile al parametro TypeOtpAuth)
- **tsa_identity** = Oggetto TSAAuth opzionale, contenente i dati per l'autenticazione Al Server TSS. Se non specificato il server userà l'account di default se configurato.
- **user** = Stringa contenente la user dell'utenza di marcatura temporale
- **password** = Stringa contenente la password dell'utenza di marcatura temporale
- **notifymail** = Stringa utilizzata per specificare l'indirizzo mail a cui notificare il completamento dell'operazione di firma.
- **notify_id** = Stringa contenente l'id che l'utente intende associare alla sessione di firma per le finalità di notifica del completamento dell'operazione. Utilizzata esclusivamente nel caso di Transport = DIRECTORYNAME. Nel caso di firma di singolo file questa proprietà può essere omessa ovvero valorizzata a NULL
- **certID** = RFU. Valorizzare con AS0
- **profile** = RFU. Impostare a NULL

OUTPUT

Oggetto SignReturnV2 contenente i seguenti parametri:

- **status** = Stringa contenente l'esito dell'operazione di firma
    - OK -> Operazione effettuata correttamente
    - Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno delle property SignReturnV2.description e SignReturnV2.return_code.
- **binaryoutput** = Byte array contenente il file firmato
- **stream** = DataHandler per scaricare file di grandi dimensioni in formato Stream. Utilizzato esclusivamente nel caso di Transport = STREAM. (NOTA: La soluzione e disponibile solo con client JAX-WS compliant.)
- **dstPath** = Stringa contenente il percorso specificato all'interno del parametro SignRequestV2.DstName in fase di richiesta. Nel caso in cui sia il file specificato in SignRequestV2.DstName già esista la property SignReturnV2.DstPath contiene il percorso al file effettivamente generato da ARSS.
- **return_code** = Enum contenente il codice di errore
    - "0001" -> Errore generico nel processo di firma
    - "0002" -> Parametri non corretti per il tipo di trasporto indicato
    - "0003" -> Errore in fase di verifica delle credenziali
    - "0004" -> Errore nel PIN (maggiori dettagli su SignReturnV2.description)
    - "0005" -> Tipo di trasporto non valido
    - "0006" -> Tipo di trasporto non autorizzato
    - "0008" -> Impossibile completare l'operazione di marcatura temporale (es irraggiungibilità del servizio, marche residue terminate, etc..)
    - "0009" -> Credenziali di delega non valide
    - "0010" -> Lo stato dell'utente non è valido (es. utente sospeso)
- **description** = Stringa contenente una descrizione di alto livello dell'errore riscontrato e/o eventuali dettagli relativi ai codici ritornati all'interno della property SignReturnV2.return_code

Utility per il recupero degli utenti all'interno di un dominio

Il metodo di utilità da utilizzare per il recupero degli utenti all'interno di un dominio è

```
public CredentialListReturn credentials_query(CredentialListQuery credential_query)
```

La ricerca è di tipo puntuale

INPUT

- **credentials_query** - Oggetto CredentialListQuery contenente i seguenti parametri
  - **constraints** = Lista di QueryConstraint contenenti i seguenti parametri
    - **field** = Stringa contenente il nome dell'attributo ricercato (unico valore ammesso al momento CF)
    - **value** = Stringa contenente il valore dell'attributo ricercato
  - **appidentity** = Oggetto ApplicationAuth contenente le credenziali applicative
  - **domain**= Stringa contenente il dominio

OUTPUT

Oggetto CredentialListReturn contenente i seguenti parametri:

- **credentials** = Lista di CredentialInfo contenente i seguenti parametri
  - **userid** = Stringa contenente l'id dell'utente
  - **certs** = Array di oggetti Certificato a disposizione dell'utente
  - **status** = Stringa contenente lo stato dell'utente, uno tra:
    - **V** = valido
    - **S** = sospeso
    - **C** = cancellato

Allegati

- **ARSS_javadoc.zip:**

Javadoc di ARSS

- **ArubaSignService.xml**

WSDL descrittivo del formato dei messaggi SOAP utilizzati da ARSS

- **ArubaSignService.xsd**

Schema descrittivo degli oggetti utilizzati dal protocollo implementato da ARSS.

Credential Proxy

Developer's Guide

ARSS Versione 2.5.0

Revisione: 29 June 2018

Aruba PEC S.p.A.
VIA SAN CLEMENTE 53
24036 PONTE SAN PIETRO (BG)
Tel. +39 0575.050.350 - FAX +39 0575.862.350
www.pec.it
an aruba.it company

Capitale Sociale € 6.500.000,00 i.v.
REA: 445886
Codice Fiscale 01879020517
Partita IVA 01879020517
R.E.A. 436479

Gestione Credential Proxy

Abilitazione Credential Proxy

Per abilitare il modulo Credential Proxy in ARSS è necessario effettuare una delle seguenti operazioni

1. Abilitazione da pagina web
    - Collegarsi alla pagina https://HOSTNAME/ArubaSignService/admin/credentialProxyConfig.xhtml
    - Autenticarsi con le credenziali amministrative.
    - Inserire la password del keystore e premere il pulsante "Abilita" (Tale operazione effettuerà la creazione del keystore e l'avvio del modulo, se il keystore non esiste. Altrimenti verificherà la password rispetto al keystore esistente per l'avvio del modulo)

2. Abilitazione da servizio SOAP/REST

    - Invocare il metodo "enable" del servizio https://HOSTNAME/ArubaSignService/CredentialProxyService con i parametri relativi alle crednziali amministrative configurate al punto 1 e la password del keystore

Ad ogni riavvio dell'applicativo sarà necessario effettuare l'abilitazione del modulo, in quanto la password del keystore è conosciuta esclusivamente dall'amministratore

Verifica Stato Credential Proxy

Per la verifica del corretto funzionamento e attivazione del modulo Credential Proxy sono possibili due modalità:

1. Verifica stato servizio SOAP/REST

    Per la verifica dello stato tramite WS è necessario invocare il metodo "status" del servizio https://HOSTNAME/ArubaSignService/CredentialProxyService che risponde con un oggetto di tipo statusResponse al cui interno la property "return"

2. Verifica stato tramite HTTP GET

    Per la verifica dello stato tramite chiamata HTTP GET è necessario richiamare la seguente URL : https://HOSTNAME/ArubaSignService/credentialproxytest che risponderà con la seguente stringa:
    - DISABLED (Se il modulo del Credential Proxy è disattivato)
    - ENABLED (Se il modulo del Credential Prodxy è attivo)

Utilizzo del modulo Credential Proxy

Per indicare ad ARSS di utilizzare il modulo Credential Proxy in fase di utilizzo dei servizi di firma, è necessario specificare il valore "$credential_proxy" nel campo userPWD dell'oggetto Identity.
In questo modo ARSS richiederà la password al modulo Credential Proxy ed eseguirà il metodo di firma con la password recuperata.

Nel caso in cui venga utilizzata tale metodologia, ARSS potrà ritornare due errori specifici per indicare delle anomalie conosciute:

- 1000 (Indica che il credential proxy non è attivo)
- 1001 (indica che il credential proxy è attivo, ma l'utente non ha alcuna password memorizzata)

Credential Proxy Services

Stato del Modulo Credential Proxy

Il metodo da utilizzare per sapere se il modulo è attivo o disattivo è status(). Il metodo è così definito.

```
public String status()
```

INPUT
n/a

OUTPUT
Oggetto statusResponse contenente la stringa relativo allo stato corrente, la quale può assumere i seguenti valori:

- enabled
- disabled

Abilitazione del Modulo Credential Proxy

Il metodo da utilizzare per abilitare il Credential Proxy è enable(). Il metodo è così definito.

```
public ArssReturn enable(CredentialProxyAdmin admin, String keystorePwd)
```

INPUT

admin - Oggetto CredentialProxyAdmin che contenente i dati per l'autenticazione dell'amministatore, così definiti:

- **adminUser** = Stringa contenente lo username dell'amministatore
- **adminPwd** = Stringa contenente la password dell'amministatore

keystorePwd = Stringa contenente la password del keystore di tipo P12 utilizzato per cifrare/decifrare le password memorizzate su filesystem

OUTPUT
Oggetto enableResponse contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata

- OK -> Operazione eseguita con successo
- Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property description.

description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno (4 cifre numeriche) del servizio

- 0000 → Operazione eseguita con successo
- Diverso da 0000 → Codice relativo al'errore riscontrato durante l'operazione

Disabilitazione del Modulo Credential Proxy

Il metodo da utilizzare per disabilitare il Credential Proxy è disable(). Il metodo è così definito.

```
public ArssReturn disable(CredentialProxyAdmin admin)
```

INPUT
admin - Oggetto CredentialProxyAdmin che contenente i dati per l'autenticazione dell'amministatore, così definiti:

- **adminUser** = Stringa contenente lo username dell'amministatore
- **adminPwd** = Stringa contenente la password dell'amministatore

OUTPUT
Oggetto disableResponse contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata

- OK -> Operazione eseguita con successo

- Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property description.

description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno (4 cifre numeriche) del servizio

- 0000 → Operazione eseguita con successo
- Diverso da 0000 → Codice relativo al'errore riscontrato durante l'operazione

Verifica della presenza della credenziale memorizzata nel Credential Proxy per uno specifico utente

Il metodo da utilizzare per sapere se uno specifico utente ha la propria credenziale memorizzata sul Credential Proxy è existingCredential(). Il metodo è così definito.

```
public ArssReturn existingCredentiai(CredentialProxyAdmin admin, String username, String domain)
```

INPUT
admin - Oggetto CredentialProxyAdmin che contenente i dati per l'autenticazione dell'amministatore, così definiti:

- **adminUser** = Stringa contenente lo username dell'amministatore
- **adminPwd** = Stringa contenente la password dell'amministatore

username = Stringa contenente lo username dell'utente
domain = Stringa contenente la password dell'utente

OUTPUT
Oggetto disableResponse contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata

- OK -> Operazione eseguita con successo
- Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property description.

description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno (4 cifre numeriche) del servizio

- 0000 → Operazione eseguita con successo

- Diverso da 0000 → Codice relativo all'errore riscontrato durante l'operazione

Memorizzazione della credenziale nel Credential Proxy per uno specifico utente

Il metodo da utilizzare per memorizzare la credenziale di uno specifico utente sul Credential Proxy è setCredential().
Il metodo è così definito.

```
public ArssReturn setCredentiai(CredentialProxyAdmin admin, String username, String domain, String password)
```

INPUT
admin - Oggetto CredentialProxyAdmin che contenente i dati per l'autenticazione dell'amministatore, così definiti:

- **adminUser** = Stringa contenente lo username dell'amministatore
- **adminPwd** = Stringa contenente la password dell'amministatore

username = Stringa contenente lo username dell'utente
domain = Stringa contenente il dominio dell'utente
password = Stringa contenente la password dell'utente

OUTPUT
Oggetto disableResponse contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata

- OK -> Operazione eseguita con successo
- Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property description.

description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno (4 cifre numeriche) del servizio

- 0000 → Operazione eseguita con successo
- Diverso da 0000 → Codice relativo all'errore riscontrato durante l'operazione

Cancellazione della credenziale memorizzata nel Credential Proxy per uno specifico utente

Il metodo da utilizzare per cancellare la credenziale di uno specifico utente memorizzata sul Credential Proxy è removeCredential().
Il metodo è così definito.

public ArssReturn removeCredentiai(CredentialProxyAdmin admin, String username, String domain)

INPUT

admin - Oggetto CredentialProxyAdmin che contenente i dati per l'autenticazione dell'amministatore, così definiti:

- **adminUser** = Stringa contenente lo username dell'amministatore
- **adminPwd** = Stringa contenente la password dell'amministatore

username = Stringa contenente lo username dell'utente
domain = Stringa contenente il dominio dell'utente

OUTPUT

Oggetto disableResponse contenente i seguenti parametri:
status = Stringa contenente l'esito della ricerca effettuata

- OK -> Operazione eseguita con successo
- Diverso da OK -> In caso di errore. Ulteriori dettagli sono riportati all'interno della property description.

description = Stringa contenente una descrizione di alto livello dell'errore riscontrato in fase di ricerca dei certificati associati all'utente
return_code = Stringa contenente il codice di ritorno (4 cifre numeriche) del servizio

- 0000 → Operazione eseguita con successo
- Diverso da 0000 → Codice relativo all'errore riscontrato durante l'operazione

Allegati

- **ARSS_2.5.X_doc.zip:** apidocs di ARSS e Credential Proxy

Integrazione Firma Remota

Ambiente demo Aruba

Il seguente documento ha lo scopo di fornire informazioni utili all'integrazione della Firma Remota tramite l'ambiente di demo Aruba pubblicamente accessibile via Internet

Url del servizio:

wsdl ARSS Demo: **https://arss.demo.firma-automatica.it/ArubaSignService/ArubaSignService?wsdl**

xsd ARSS Demo: **https://arss.demo.firma-automatica.it/ArubaSignService/ArubaSignService?xsd=1**

NB: In ambiente di demo l'utenza: "`titolare_rem`" è stata settata in modo tale da non potersi bloccare. Vi ricordiamo che in ambiente di produzione dopo 4 tentativi di inserimento OTP errato l'utenza viene bloccata

Credenziali per ambiente Demo:

```
<identity>
          <otpPwd>0973487814</otpPwd> (anche per opensession)
          <typeOtpAuth>demoprod</typeOtpAuth>
          <user>titolare_rem</user>
          <userPWD>password22</userPWD>
</identity>
```

Esempio di pkcs7signV2 firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns1:pkcs7signV2 xmlns:ns1="http://arubasignservice.arubapec.it/">
            <SignRequestV2>
                <certID>AS0</certID>
                <identity>
                    <otpPwd>0973487814</otpPwd>
                    <typeOtpAuth>demoprod</typeOtpAuth>
                    <user>titolare_rem</user>
                    <userPWD>password22</userPWD>
                </identity>
                <requiredmark>false</requiredmark>
                <stream>RmFicml6aW87UGVyYW....XXX</stream>
                <transport>STREAM</transport>
            </SignRequestV2>
            <detached>false</detached>
            <returnder>true</returnder>
        </ns1:pkcs7signV2>
    </soapenv:Body>
</soapenv:Envelope>
```

Integrazione Firma Remota

Ambiente demo Aruba

Esempio di xmlsignature firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:arub="http://arubasignservice.arubapec.it/">
   <soapenv:Header/>
   <soapenv:Body>
      <arub:xmlsignature>
         <SignRequestV2>
            <binaryinput>PD94bWwgdm.....0aXZpPgo=</binaryinput>
            <certID>AS0</certID>
            <identity>
               <otpPwd>0973487814</otpPwd>
               <typeOtpAuth>demoprod</typeOtpAuth>
               <user>titolare_rem</user>
               <userPWD>password22</userPWD>
             </identity>
             <requiredmark>false</requiredmark>
             <transport>BYNARYNET</transport>
         </SignRequestV2>
          <parameter>
          <type>XMLENVELOPED</type>
         </parameter>
      </arub:xmlsignature>
   </soapenv:Body>
</soapenv:Envelope>
```

Esempio di pdfsignatureV2 firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:arub="http://arubasignservice.arubapec.it/">
   <soapenv:Header/>
   <soapenv:Body>
      <arub:pdfsignatureV2>
         <SignRequestV2>
            <binaryinput>JVBERi0xL.....U9GDQo=</binaryinput>
            <certID>AS0</certID>
             <identity>
                    <otpPwd>0973487814</otpPwd>
                    <typeOtpAuth>demoprod</typeOtpAuth>
                    <user>titolare_rem</user>
                    <userPWD>password22</userPWD>
                </identity>
            <requiredmark>true</requiredmark>
            <transport>BYNARYNET</transport>
         </SignRequestV2>
      </arub:pdfsignatureV2>
   </soapenv:Body>
</soapenv:Envelope>
```

Integrazione Firma Remota

Ambiente demo Aruba

Esempio di opensession firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:arub="http://arubasignservice.arubapec.it/">
   <soapenv:Header/>
   <soapenv:Body>
      <arub:opensession>
         <Identity>
            <otpPwd>0973487814</otpPwd>
            <typeOtpAuth>demoprod</typeOtpAuth>
            <user>titolare_rem</user>
            <userPWD>password22</userPWD>
         </Identity>
      </arub:opensession>
   </soapenv:Body>
</soapenv:Envelope>
```

Esempio di pkcs7signV2 firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns1:pkcs7signV2 xmlns:ns1="http://arubasignservice.arubapec.it/">
            <SignRequestV2>
                <certID>AS0</certID>
                <identity>
                    <typeOtpAuth>demoprod</typeOtpAuth>
                    <user>titolare_rem</user>
                    <userPWD>password22</userPWD>
                </identity>
                <session_id>RETURN OPEN SESSION</session_id>
                <requiredmark>false</requiredmark>
                <stream>RmFicml6aW87UGVyYW....XXX</stream>
                <transport>STREAM</transport>
            </SignRequestV2>
            <detached>false</detached>
            <returnder>true</returnder>
        </ns1:pkcs7signV2>
    </soapenv:Body>
</soapenv:Envelope>
```

Integrazione Firma Remota

Ambiente demo Aruba

Esempio di closesession firma Remota per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:arub="http://arubasignservice.arubapec.it/">
   <soapenv:Header/>
   <soapenv:Body>
      <arub:closesession>
         <Identity>
            <otpPwd>0973487814</otpPwd>
            <typeOtpAuth>demoprod</typeOtpAuth>
            <user>titolare_rem</user>
            <userPWD>password22</userPWD>
         </Identity>
         <sessionid> RETURN OPEN SESSION </sessionid>
      </arub:closesession>
   </soapenv:Body>
</soapenv:Envelope>
```

Esempio di marcatura temporale (tsr) per ambiente Demo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:arub="http://arubasignservice.arubapec.it/">
   <soapenv:Header/>
   <soapenv:Body>
      <arub:tsr>
         <MarkRequest>
            <user>test01</user>
            <password>test01</password>
            <transport>BYNARYNET</transport>
            <binaryinput>RmFicml6aW87UGVyYW....XXX</binaryinput>
         </MarkRequest>
      </arub:tsr>
   </soapenv:Body>
</soapenv:Envelope>
```

Integrazione Firma Remota

Ambiente demo Aruba

Credenziali Certificato Sigillo per ambiente Demo:

```
<identity>
            <otpPwd>dsign</otpPwd> per firma
            <typeOtpAuth>demoprod</typeOtpAuth>
            <user>demosigillo</user>
            <userPWD>password22</userPWD>
</identity>
```