Arkitekturbeskrivning av SDK-lösning med Hubs-meddelandeklient

1. Inledning

Denna tekniska bilaga beskriver en **säker digital kommunikation (SDK)**-lösning där *meddelandeklienten (MK)* är implementerad som en app i en Hubs-instans. Syftet är att ge en översiktlig, topologisk systembeskrivning över hur meddelanden skickas och tas emot genom systemet – från en avsändaranvändare via hela infrastrukturen till en mottagaranvändare. Beskrivningen tar upp arkitektur, ingående komponenter, kommunikationsflöden, protokoll/gränssnitt, säkerhetsaspekter samt drifts- och redundansfrågor.

Lösningen är tänkt att uppfylla kraven i federationen **Säker digital kommunikation (SDK)**, vilken möjliggör enhetlig och säker överföring av känslig information mellan organisationer i offentlig sektor. SDK-ramverket bygger på standarder (EU CEF eDelivery) och gemensamma regelverk, vilket innebär att deltagande system kan kommunicera via gemensamma protokoll och en betrodd infrastruktur. I detta scenario fungerar Hubs-applikationen som användarens gränssnitt (meddelandeklient) för att skicka och läsa säkra meddelanden, medan bakomliggande tjänster sköter transport, kryptering och leverans i enlighet med SDK.

Den fullständiga kedjan som beskrivs omfattar:

- **Meddelandeklient (MK) i Hubs** – där avsändaren skriver och skickar meddelandet.
- **Intern mailserver/omvandlingskomponent (Mailgateway)** – som via SMTP tar emot meddelandet från MK och omvandlar det till ett API-anrop.
- **Meddelandetjänst (MT)** – organisationens meddelandeväxel som via SDK API tar emot meddelandet från mailgateway, krypterar/signerar och hanterar vidarebefordran.
- **Accesspunkt (AP)** – transportnod som tar emot meddelandet från MT och skickar det vidare över SDK-federationens nätverk (via internet) till mottagarens accesspunkt.
- **Mottagarens AP** – tar emot meddelandet inom federationen och gör det tillgängligt för mottagarens system.
- **Mottagarens MT** – tar emot meddelandet från sin AP, dekrypterar/verifierar och levererar det till rätt meddelandeklient.

- **Mottagarens MK (Hubs)** – där mottagaren notifieras om det inkomna meddelandet och kan läsa det i sin meddelandeklient.

Observera att denna beskrivning beskriver ett system uppsatt utan Confidential compute. I det fallet när Hubs är konfigurerat med confidential compute tillkommer krypteringssteg vid uppstart av miljön. Detta beskrivs i detalj i dokumentet ”Teknisk referensarkitektur” som finns med som bilaga till detta dokument

2. Översiktlig arkitektur

Översiktligt består lösningen av tre huvudlager: **verksamhetslagret** (meddelandeklienter och användare), **meddelandelagret** (meddelandetjänster och eventuella integrationer) och **transportlagret** (accesspunkter och nätverksinfrastruktur). Figur 1 nedan visar topologin för hur komponenterna hos avsändarorganisation A samverkar med mottagarorganisation B genom SDK-federationen.

- **Avsändarens Hubs (MK)** innehåller en meddelandeklient-app som utgör användarens gränssnitt för säkra meddelanden. Användaren skriver ett meddelande och när det skickas går det via **SMTP** till en **intern mailgateway**.

- **Mailgateway** är en kombinerad mailserver/omvandlingskomponent internt hos organisationen. Den tar emot meddelandet som ett e-postmeddelande och översätter det till ett anrop mot meddelandetjänstens API (enligt SDK-gränssnitt D).

- **Meddelandetjänsten (MT)** är en serverkomponent som hanterar själva meddelandeöverföringen. Den tar emot anropet från mailgateway (med meddelandets innehåll, adressat m.m.), utför nödvändiga säkerhetsåtgärder (t.ex. kryptering och signering av meddelandet) och lämnar över meddelandet till organisationens **accesspunkt (AP)** för transport.

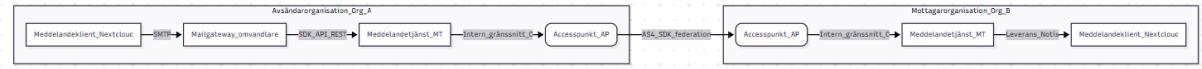
- **Accesspunkten (AP)** är organisationens nod i SDK-nätverket. AP:n ansvarar för att genomföra den faktiska transporten över internet till mottagarens accesspunkt via det gemensamma protokollet (AS4) samt att säkerställa säkert utbyte och kvittens på transportnivå.

- **SDK-federationen** (nätverket) utgörs av alla anslutna accesspunkter och gemensamma tjänster. Meddelandet routas från avsändarens AP till mottagarens AP baserat på adressinformation och metadata som finns publicerade i den gemensamma infrastrukturen (t.ex. SDK Adressbok och SMP).

- **Mottagarens AP** tar emot meddelandet och levererar det vidare internt till **mottagarens meddelandetjänst (MT)** via ett lokalt gränssnitt.

- **Mottagarens MT** dekrypterar meddelandet, kontrollerar signaturer och validerar innehållet. Därefter placeras meddelandet i mottagarens meddelandeklient (t.ex. genom att MT notifierar klienten eller tillåter den hämta meddelandet via API).

- **Mottagarens MK (Hubs)** visar slutligen det inkomna meddelandet för användaren på mottagarsidan. Mottagaren kan läsa meddelandet i sin Hubs-applikation och eventuellt svara, varpå ett omvänt flöde tar vid.



Figur 1: Topologisk översikt över SDK-lösningen. Diagrammet visar de ingående komponenterna hos avsändare (vänster) och mottagare (höger), samt kommunikationen dem emellan. Avsändarens meddelandeklient (MK) i Hubs kommunicerar internt med en mailgateway via SMTP. Mailgateway:n anropar meddelandetjänsten (MT) via ett REST-baserat SDK API. Meddelandetjänsten överför meddelandet till den lokala accesspunkten (AP), som i sin tur skickar meddelandet via AS4-protokollet genom SDK-federationens nätverk till mottagarens accesspunkt. Hos mottagaren sker motsvarande process omvänt: accesspunkten lämnar meddelandet till mottagarens meddelandetjänst, som gör det tillgängligt för mottagarens meddelandeklient i Hubs.

Arkitekturen följer SDK:s referensmodell, där meddelandeklient, meddelandetjänst och accesspunkt utgör separata logiska komponenter med tydliga roller . Detta möjliggör att olika leverantörers produkter kan användas för respektive komponent, så länge de följer standardgränssnitten och reglerna i federationen. I den gemensamma infrastrukturen finns också stödtjänster som adressregister och certifikathantering som används vid adressuppslag och nyckelutbyte, vilket nämns senare i rapporten.

3. Komponentbeskrivningar

Här beskrivs de fyra huvudsakliga komponenterna i lösningen mer detaljerat: Meddelandeklient (MK), Mailgateway, Meddelandetjänst (MT) och Accesspunkt (AP).

3.1 Meddelandeklient (MK)

Meddelandeklienten är det gränssnitt som användaren direkt interagerar med för att skicka och ta emot säkra meddelanden. I denna lösning är MK implementerad som en app i en Hubs-plattform, vilket innebär att användaren når den via sin vanliga Hubs-inloggning i webbläsaren konfigurerad med LOA3 eller vad organisationen väljer att konfigurera det med. MK-appen erbjuder funktionalitet liknande en säker e-postbrevlåda: användaren kan komponera nya meddelanden, bifoga filer, ange mottagare samt läsa inkommande meddelanden i en inkorg.

Meddelandeklienten hanterar bland annat koncept som **brevlådor** (inkorg, utkorg), visning av meddelandetrådar och möjlighet att besvara meddelanden i en säkert hanterad miljö. Klienten ansvarar också för att sätta rätt **adressat** på meddelandet (t.ex. välja organisation och funktion hos mottagaren, ofta via uppslag i en adressbokstjänst) samt för att lagra skickade och mottagna meddelanden på lämpligt sätt. I Hubs-appen kan lagring ske i databasen eller krypterat i filområdet, beroende på implementation – det viktiga är att endast behöriga användare kommer åt innehållet.

För att skicka ett meddelande initierar användaren processen via MK:s gränssnitt (t.ex. klickar på "Skicka"). Hubs-appen paketerar då meddelandets innehåll, metadata (som mottagarens adress, ärendemening etc) och eventuella bilagor. Eftersom MK i detta fall inte direkt pratar med meddelandetjänstens API, utan använder e-post som transport internt, skickar appen meddelandet vidare via SMTP till en intern gateway inuti hubs

(se nedan). Detta kan ske genom att Hubs använder en konfigurerad SMTP-server (mailgateway-komponenten) som mottagare för utgående meddelanden.

Meddelandeklienten kräver **användarautentisering**, vilket i Hubs sker via plattformens vanliga mekanismer (t.ex. användarnamn/lösenord, tvåfaktorsautentisering, SSO eller BankID-integration beroende på hur Hubs är uppsatt). Endast behöriga användare kan komma åt MK-appen och därmed läsa/skicka SDK-meddelanden. MK i Hubs kan också logga relevanta händelser (som att ett meddelande skickats eller öppnats) för spårbarhet, i enlighet med säkerhetsmodellen (se avsnitt 6).

Sammanfattningsvis är meddelandeklienten Hubs-appens roll att vara en användarvänlig och säker yta för meddelandeutbyte. Den döljer den underliggande komplexiteten och presenterar meddelanden och kvittenser i en begriplig form för användaren. MK representerar också avsändarens eller mottagarens **funktionsbrevlåda** (om meddelanden skickas till en funktionsadress snarare än en enskild person) genom att associera till rätt funktionscertifikat och adress.

3.2 Mailgateway (intern mailserver/omvandlingskomponent)

Mailgateway-komponenten fungerar som en brygga mellan Hubs-miljön (som hanterar meddelandet som e-post) och meddelandetjänsten (som använder ett REST/JSON-baserat SDK-API). Den kan ses som en intern *omvandlare* eller *adapter*. På ena sidan exponerar den en **SMTP-server** som Hubs/MK skickar meddelandet till, och på andra sidan agerar den klient mot meddelandetjänstens API.

När användaren skickar ett meddelande från MK kommer det alltså in till mailgatewayn som ett e-postmeddelande. Gatewayn är vanligtvis konfigurerad som en intern SMTP-tjänst (t.ex. på en specifik domän eller port som Hubs känner till). E-postmeddelandet innehåller all nödvändig information: mottagaradress (förmodligen i form av en speciell adress som identifierar mottagarens **SDK-adress** eller funktionsbrevlåda), avsändarinformation, ämne och meddelandetext samt bilagor.

Omvandling till API-anrop: Mailgatewayn tolkar det inkomna e-postmeddelandet och översätter det till ett **SDK API-anrop** mot meddelandetjänsten. SDK har ett definierat API (gränssnitt D) mellan meddelandeklient och meddelandetjänst för bl.a. att skicka och hämta meddelanden. I detta fall agerar mailgatewayn som en automatisk klient gentemot det API:et. Den plockar ut relevanta fält ur e-postet – till exempel mottagarens organisations-ID eller adress, funktionsadress/ämneskategori, meddelandets textinnehåll och eventuella bilagor – och anropar meddelandetjänstens *”skicka meddelande”*-funktion via HTTP(S) (REST). Detta anrop överför meddelandets innehåll och metadata i JSON- eller XML-format enligt API-specifikationen.

Mailgatewayn behöver ha nödvändiga autentiseringsuppgifter för att få anropa meddelandetjänstens API (t.ex. en API-nyckel, token eller certifikat för klientautentisering, beroende på hur API:et är skyddat). Gatewayn kan också utföra grundläggande kontroll, t.ex. att avsändande användare är behörig och att mottagaradressen är i rätt format. I vissa implementationer kan mailgatewayn integrera med SDK Adressbok för att slå upp mottagare innan meddelandet skickas, men oftast antar vi att MK redan angivit en giltig adress.

3.3 Meddelandetjänst (MT)

Meddelandetjänsten utgör kärnan i organisationens säkra meddelandehantering. Den kan beskrivas som en meddelandeväxel eller mellanskikt som tar emot meddelanden från interna klienter och levererar dem via transportinfrastrukturen, och vice versa. I SDK-arkitekturen ansvarar meddelandetjänsten för en rad kritiska funktioner:

- **Kryptering och signering:** Innan ett meddelande skickas ut ur organisationen krypterar meddelandetjänsten innehållet med mottagarorganisationens publika nyckel (så kallad end-to-end-kryptering mellan avsändar- och mottagarorganisation, även kallat O2O-kryptering) och signerar meddelandet digitalt med avsändarorganisationens privata nyckel. Detta säkerställer att endast rätt mottagare kan läsa innehållet och att mottagaren kan verifiera avsändarens identitet samt att innehållet inte har manipulerats. (Certifikathanteringen för detta beskrivs mer i säkerhetsavsnittet.)

- **Dekryptering och validering:** Motsvarande funktion vid mottagning – när ett inkommande meddelande hämtas från accesspunkten dekrypterar meddelandetjänsten innehållet (med den egna privata nyckeln) och verifierar avsändarens signatur samt integriteten. Om något av detta misslyckas kan meddelandet avvisas.

- **Kommunikation med accesspunkt:** MT hanterar att lämna utgående meddelanden till organisationens AP för transport, samt att hämta inkommande meddelanden från AP:n. Gränssnittet mellan MT och AP kallas *gränssnitt C* i SDK och är inte strikt standardiserat (det kan variera beroende på leverantör). I praktiken erbjuder många AP-programvaror någon form av API eller kö-mekanism för MT. Exempelvis kan AP:n exponera en lokal **REST- eller JMS-baserad tjänst** som MT använder för att överföra meddelandet. I vår lösning är meddelandetjänsten ansluten till AP:n genom en sådan intern integration – förmodligen en säker anrop inom samma nät eller servermiljö – för att lägga meddelandet i AP:ns utgående kö eller hämta från dess inkommande kö.

- **Validering av meddelanden:** MT kan utföra affärslogiska valideringar på meddelandet innan det skickas eller efter att det tagits emot. Till exempel kontrollera att obligatoriska metadata finns, att innehållsformat är korrekt (t.ex. om meddelandet har en struktur som ska följas), och att mottagarens adress faktiskt existerar i SDK-adressboken. Om något är fel kan MT avbryta sändningen och generera ett felmeddelande tillbaka till avsändande klient.

- **Adressuppslagning:** Innan sändning behöver meddelandetjänsten översätta mottagarens funktionsadress eller organisations-ID till tekniska detaljer för leverans. Detta görs via SDK:s gemensamma adress- och metadata-tjänster. MT använder *SDK Adressbok* och *SMP (Service Metadata Publisher)* för att slå upp mottagarorganisationens uppgifter: t.ex. vilken accesspunkt de använder, adressen/URL:en till den accesspunkten, vilken typ av meddelandeformat de stöder och mottagarens publika krypteringsnyckel. Dessa data behövs för att paketera meddelandet korrekt för transport och kryptera det. (Adressboken nås via gränssnitt B i SDK som är ett separat REST-API tillhandahållet av t.ex. Digg).

- **Köhantering och asynkronitet:** Meddelandetjänsten implementerar köer för utgående och inkommande meddelanden, vilket möjliggör asynkron överföring. När MT lämnar ett meddelande till AP behöver den inte vänta på slutlig leverans,

utan kan direkt gå vidare – AP:n meddelar senare utfallet. På motsvarande sätt kan inkommande meddelanden ligga i kö om mottagarens MK (t.ex. Hubs) inte är redo omedelbart. Denna asynkrona design ökar robustheten: om någon komponent är tillfälligt nere, ligger meddelanden kvar i köerna tills de kan levereras.

- **Intern routing:** Om organisationen har flera olika meddelandeklienter eller funktionsbrevlådor, kan meddelandetjänsten utifrån adressfältet avgöra till vilken intern brevlåda eller applikation ett inkommande meddelande ska levereras. I vår enkla scenario kanske allting går till Hubs-MK, men i en större organisation kan samma meddelandetjänst betjäna flera system.

- **Statushantering och kvittenser:** MT tar emot leveranskvittenser och status från transportlagret (AP) och från mottagarens sida och uppdaterar status för skickade meddelanden. När mottagarens system har tagit emot och godkänt meddelandet genereras en **meddelandekvittens** (mjukvarumässig leveransbekräftelse) tillbaka till avsändaren. Denna kvittens tas emot av avsändarens meddelandetjänst och förmedlas vidare till MK så att avsändaren kan se att meddelandet nått fram. Om kvittensen indikerar fel (t.ex. *REJECTED*), eller om ingen kvittens kommer inom viss tid, kan MT flagga detta och informera avsändarens MK (t.ex. genom en notis om leveransproblem). Även transportkvittens på AS4-nivå hanteras av MT/AP tillsammans, men dessa är mer tekniska (t.ex. mottagande AP bekräftar att den fått paketet).

Meddelandetjänsten kan implementeras som en fristående serverapplikation (on-premises eller molnbaserad) levererad av någon SDK-leverantör, eller som en modul integrerad i accesspunktsprogramvaran. Det viktiga är att den uppfyller SDK:s krav och eventuella certifieringar. Flera leverantörer (t.ex. Inera, Digg m.fl.) har tagit fram referensimplementationer eller guidelines för meddelandetjänster och hur de ska samverka med MK och AP. Vår meddelandetjänst följer *SDK API MT/MK*-specifikationen på gränssnitt D, vilket innebär att den exponerar ett standardiserat REST-API för klienter. Detta gör det möjligt att potentiellt ansluta andra klientapplikationer eller verksamhetssystem i framtiden utan specialanpassning, så länge de också stödjer SDK API:et.

3.4 Accesspunkt (AP)

Accesspunkten är den komponent som hanterar den *federerade transporten* av meddelanden mellan organisationer. Varje organisation som deltar i SDK-nätverket har (eller anlitar) en accesspunkt som knyter ihop den egna infrastrukturen med det gemensamma nätverket. Accesspunkten kan liknas vid en säker digital *postterminal* som sköter transportkuvertering, vägval och leveranskvittens på transportnivå.

I denna lösning har både avsändarorganisation (Org A) och mottagarorganisation (Org B) var sin AP. Accesspunkternas viktigaste egenskaper och ansvar är:

- **Standardiserad kommunikation över gräns A:** AP kommunicerar med andra accesspunkter enligt ett gemensamt protokoll. SDK använder *OASIS ebMS 3.0 AS4* som transportprotokoll för att utbyta meddelanden mellan accesspunkter. AS4 är en webbtjänst-baserad meddelandeprotokollstandard som möjliggör säkra, kvitterade leveranser oberoende av nyttolastens format. Genom att följa samma profil av AS4 (enligt CEF eDelivery och Digg:s specifikationer) kan alla

accesspunkter i nätverket interoperera med varandra utan särskilda anpassningar – detta är en förutsättning för SDK-federationen.

- **Teknisk adressering och routing:** När meddelandetjänsten lämnar över ett meddelande till AP för leverans, är det AP:ns uppgift att lista ut *vilken fjärr-AP* som ska ta emot det. Detta baseras på mottagarens adresseringsmetadata. AP använder information från den gemensamma **SMP (Service Metadata Publisher)** för att slå upp mottagarorganisationens accesspunkt-adress (URL) samt eventuella specifika parametrar för överföringen. SMP:en är en katalog där varje organisation publicerar sina mottagartjänster, tekniska endpoint-adresser och certifika. När AP vet vart meddelandet ska skickas, används nätverkets DNS/routing för att nå rätt mottagare.

- **Transportskydd och e-stämpling:** AP:n säkerställer att överföringen över internet är krypterad och integritetsskyddad. Detta sker dels via **TLS** (ofta med klientcertifikat för att motpartens AP ska kunna verifieras) och dels via AS4-protokollets egna säkerhetsmekanismer. AS4-meddelandet i sig signeras digitalt av avsändande AP och krypteras för mottagande AP. Varje meddelande får en unik ID och förses med tidsstämplar och eventuella spårbarhetsmetadata (e-stämpel). Mottagande AP kontrollerar signaturen och certifikatet för att se att meddelandet kommer från en betrodd avsändar-AP och att det inte förvanskats under transporten. Detta utgör en *transportkvittens* – AP B skickar en teknisk ACK tillbaka till AP A när meddelandet är mottaget korrekt.

- **Felhantering och återförsändning:** Om en leverans inte lyckas (t.ex. om mottagarens AP är nere), kan avsändande AP hantera återförsök enligt protokoll. AS4 har inbyggt stöd för kvittenser och felmeddelanden, så att avsändande sida får veta om transporten misslyckades. AP kan då lägga meddelandet i kö för nytt försök eller signalera till meddelandetjänsten att leverans inte kunde utföras.

- **Exponering av gränssnitt C till MT:** Accesspunkten erbjuder också ett **lokalt gränssnitt** för meddelandetjänsten (interface C, som nämnts ovan). Det är via detta gränssnitt AP tar emot meddelanden från MT för sändning, och där AP levererar inkommande meddelanden till MT. Beroende på produkt kan detta gränssnitt vara t.ex. en webbservice, en databas-drop, ett filkatalogövervakning eller meddelandekö. I vår beskrivning antar vi att AP har ett REST- eller meddelandekö-baserat API som MT integrerar med. Exempel: många organisationer använder Domibus (EU-kommissionens referensimplementation) som AP, vilken har ett plugin-ramverk för connectors mot interna system. Andra AP-mjukvaror kan kräva en fristående adapter om de saknar inbyggda API, men principen är densamma.

- **Drift hos AP-operatör:** Det är värt att nämna att vissa organisationer väljer att låta en extern **accesspunktsoperatör** drifta AP:n (t.ex. en leverantör som är godkänd för SDK). I andra fall kan AP driftsättas in-house. Oavsett vilket, så måste AP-operatören följa federations regler och upprätthålla säkerheten (inklusive hantering av de särskilda AP-certifikaten utfärdade av federationen, se säkerhet). Vår topologi-beskrivning utgår ifrån att AP är logiskt en del av organisationens system, även om den fysiskt kan ligga i molnet eller hos en partner.

Genom accesspunkterna uppnås en **lös koppling** mellan avsändare och mottagare. Ingen direkt anslutning finns parterna emellan, utan all kommunikation sköts *AP till AP*

i fyrhörningsmodellen (avsändare->avsändar-AP->mottagar-AP->mottagare). Detta innebär att så länge varje part har tillit till sitt AP och det gemensamma regelverket, kan vem som helst i nätverket skicka till vem som helst annan utan separata avtal för varje relation. Accesspunkten utgör därmed en kritisk komponent för både säkerhet och interoperabilitet.

4. Kommunikationsflöde (sekvensbeskrivning)

I detta avsnitt gås ett **end-to-end kommunikationsflöde** igenom steg för steg, från att avsändaren skapar ett meddelande i sin meddelandeklient till att mottagaren kan läsa det i sin. Flödet innefattar både de logiska stegen och vilka systemkomponenter som interagerar i sekvens. Figur 2 illustrerar samma flöde i form av ett sekvensdiagram.

4.1 Sekvensbeskrivning av meddelandesändning (Org A -> Org B):

1. **Meddelande skrivs:** Avsändaren (Användare A) loggar in i Hubs och öppnar meddelandeklient-appen (MK). Där skapar hen ett nytt meddelande, anger mottagare (t.ex. väljer en organisation och en funktionsbrevlåda eller specifik mottagare via adressboken) samt skriver meddelandetext och lägger till eventuella bilagor.

2. **Skicka från MK:** När användaren klickar "Skicka" så paketerar MK (Hubs-appen) all data och skickar iväg meddelandet via SMTP till den interna mailgatewayn. Här uppträder MK i princip som en e-postklient som skickar ett mail. Mottagaradressen kan exempelvis se ut som en speciell e-postadress som mailgatewayn lyssnar på (alternativt används SMTP-auth direkt mot gatewayn).

3. **SMTP-mottagning:** Mailgateway-komponenten tar emot det inkommande e-postmeddelandet. Den bekräftar mottagandet på SMTP-nivå gentemot MK (så MK vet att "mailet" är skickat). Mailgatewayn börjar nu bearbeta innehållet.

4. **Omvandling till API-anrop:** Mailgatewayn extraherar nödvändig information ur mailet: exempelvis att mottagaren är *Org B* med funktionsadress *X*, meddelandets rubrik och text, osv. Den anropar därefter **meddelandetjänst A** (MT_A) via det interna *SDK API*:et (gränssnitt D) för att leverera meddelandet. API-anropet sker över HTTPS (REST) och inkluderar meddelandets innehåll och metadata strukturerat (t.ex. i JSON). I anropet anger gatewayn autentisering så att MT vet att anropet kommer från en betrodd intern komponent.

5. **MT tar emot meddelandet:** Meddelandetjänst A tar emot API-anropet och registrerar ett nytt utgående meddelande. Den validerar först meddelandet (innehåll och mottagardata). Eventuellt görs här ett **adressuppslag** via SDK Adressbok om inte redan gjort – för att få fram tekniska leveransuppgifter för Org B. MT A hämtar troligen Org B:s mottagarinfo (inkl. certifikat) från den centrala adressboken/SMP om det inte finns cacheat.

6. **Kryptering och signering:** MT A krypterar meddelandets nyttolast med Org B:s publika krypteringscertifikat (så bara Org B kan läsa det) och signerar det

med Org A:s privata nyckel. Därmed är meddelandet förberett för säker transport på innehållsnivå.

7. **Överlämning till AP:** MT A skickar det nu krypterade och signerade meddelandet vidare till **Accesspunkt A** (AP_A) via det interna gränssnittet C. Beroende på integration kan detta vara t.ex. ett lokalt API-anrop eller att MT skriver meddelandet till en kö/databas som AP läser från. I vilket fall tar AP A emot meddelandet i sin utgående kö.

8. **Transport (AS4) till mottagar-AP:** Accesspunkt A etablerar kommunikation med **Accesspunkt B** (AP_B) över internet enligt AS4-protokollet. AP A skickar iväg meddelandet som ett AS4-meddelande, vilket inbegriper att det kuverteras i en AS4-envelope med nödvändiga header-fält, signeras av AP A och krypteras för AP B. Överföringen sker över en TLS-krypterad kanal. När AP B har mottagit meddelandet korrekt returnerar den en **transportkvittens (ack)** till AP A som bevis på att överföringen lyckades. (Om överföringen misslyckas, hanterar AP A återförsök eller felrapport till MT A.)

9. **Mottagning i AP B:** Accesspunkt B dekrypterar AS4-kuvertet (transportkrypteringen) och verifierar AP A:s signatur samt att avsändar-AP:s certifikat är betrott. Därefter kan AP B plocka ut det skyddade meddelandet (nyttolasten) och lägga det i sin inkommande kö för **Meddelandetjänst B** (MT_B). AP B notifierar MT B (t.ex. genom ett API-anrop, kömeddelande eller liknande via gränssnitt C) att ett nytt meddelande har anlänt för hämtning.

10. **Hämtning av meddelande i MT B:** Meddelandetjänst B hämtar det inkomna meddelandet från AP B:s kö. MT B dekrypterar meddelandets innehåll med sin organisations privata nyckel och verifierar avsändarsignaturen (Org A) mot avsändarens certifikat som följde med eller hämtas via certifikatregister. Om allt är okej, anses meddelandet vara *levererat* till Org B:s meddelandesystem. MT B lagrar meddelandet i sin databas och kategoriserar det enligt adressen (t.ex. vilken funktionsbrevlåda det tillhör). Den genererar omedelbart en **meddelandekvittens** (leveransbekräftelse på applikationsnivå): typiskt ett litet meddelande som säger *ACCEPTED* (eller *REJECTED* om validering misslyckades). Denna kvittens skickas asynkront tillbaka genom AP B till AP A och upp till MT A, för att signalera utfall. (Se steg 12 nedan.)

11. **Leverans till mottagarens MK:** MT B gör nu meddelandet tillgängligt för **Meddelandeklient B** (MK i Hubs hos mottagaren). Beroende på integration kan detta ske genom en push-notifiering eller att MK B kontinuerligt pollar efter nya meddelanden via API. I Hubs-appen uppdateras i alla fall mottagarens inkorg. Om realtidsnotifiering är aktiverad kan användaren B få en avisering (t.ex. via webbnotis eller e-post) om att ett nytt säkert meddelande har inkommit.

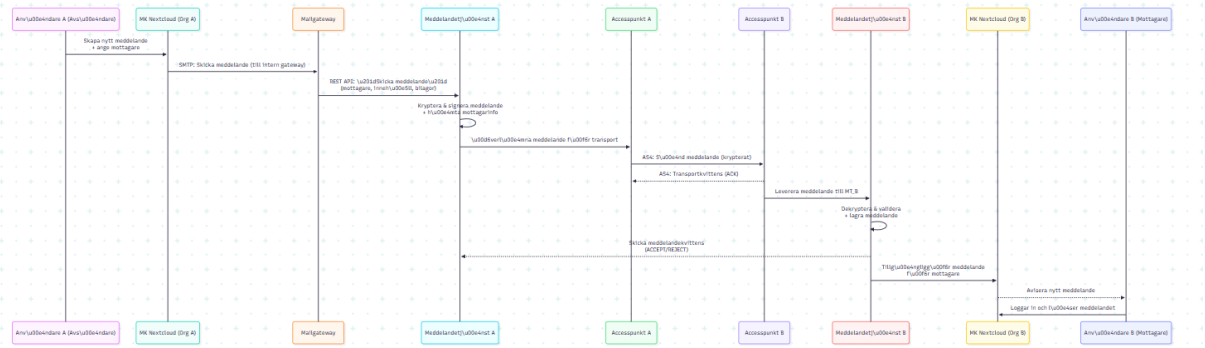
12. **Meddelandekvittens tillbaka till avsändaren:** (Detta steg sker i bakgrunden utan användarinblandning.) Efter lyckad leverans skickar MT B den tidigare nämnda meddelandekvittensen via AP B -> AP A -> MT A. Accesspunkterna utbyter denna kvittens som ett separat AS4-meddelande i motsatt riktning, vilket MT A tar emot. MT A matchar kvittensen med ursprungsmeddelandet (baserat på korrelations-ID) och markerar hos sig att meddelandet är bekräftat av mottagaren. MT A kan nu informera avsändarens MK om leveransstatus (t.ex. ändra status till *Levererat*). Om kvittensen istället signalerade *REJECTED* (t.ex.

om mottagarens MT inte kunde behandla meddelandet), markeras det som misslyckat och loggas för åtgärd.

13. **Mottagaren läser meddelandet:** Användare B blir informerad om det nya meddelandet (genom MK-gränssnittet eller notis) och loggar in i Hubs för att läsa det. MK-appen presenterar det inkomna meddelandet med dess innehåll och eventuella bilagor. Mottagaren kan nu svara på meddelandet säkert genom att skapa ett svar i MK, vilket i så fall initierar ett nytt meddelandeflöde i omvänd riktning via samma komponenter.

14. **Avslutning:** Flödet avslutas här. Avsändare A kan i sin MK se att meddelandet skickats iväg och (när kvittensen kommit) att det blivit levererat. Mottagare B har mottagit och läst meddelandet. Alla steg är loggade i respektive komponenters loggar för spårbarhet.

Nedanstående sekvensdiagram (Figur 2) illustrerar det ovan beskrivna flödet med de viktigaste interaktionerna mellan komponenterna:



Figur 2: Sekvensdiagram för meddelandeflödet från avsändare A till mottagare B.

Diagrammet visar aktörer och komponenter i tidsordning uppifrån och ner Gröna pilar representerar användarhandlingar eller informationsflöden inom en organisation, medan orange pilar representerar kommunikation över organisationsgränsen (AS4 inom SDK-federationen). Notera att meddelandekvittensen skickas tillbaka asynkront efter att mottagaren tagit emot meddelandet.

Flödet illustrerar hur varje lager samverkar: användaren interagerar med meddelandeklienten; klienten/gatewayn med meddelandetjänsten; meddelandetjänsten med accesspunkten; och accesspunkterna med varandra över internet. Den asynkrona designen med köer i MT och AP gör att systemet tål tillfälliga avbrott utan att meddelanden går förlorade – meddelanden buffras tills motparten är redo. Dessutom möjliggör kvittenserna att avsändaren får återkoppling på leveransen, vilket är viktigt ur spårbarhets- och ansvarsperspektiv i offentlig sektor.

5. Protokoll och gränssnitt

I lösningen används flera olika protokoll och gränssnitt, anpassade för sina respektive ändamål. Nedan beskrivs de viktigaste, inklusive **SMTP**, **REST** (SDK API:er) och **AS4**, samt hur de kommer till användning.

- **SMTP (Simple Mail Transfer Protocol):** Används för kommunikationen mellan meddelandeklienten i Hubs och den interna mailgatewayn. När MK skickar meddelandet, sker det i form av ett e-postmeddelande över SMTP.

Protokollet är välkänt från traditionell e-post och ger en enkel lösningsmekanism internt. Här är SMTP-sessionen begränsad till organisationens interna nät (Hubs-server till mailgateway-server), och kan köras med TLS (STARTTLS) för att kryptera överföringen trots att den är lokal. Mailgatewayn är konfigurerad som SMTP-server (MTA) och MK som SMTP-klient (MUA). Via SMTP överförs meddelandets header (inklusive From, To, Subject som kan bära adressinformation) och body/attachments (som MIME-data). SMTP ger en kvittens (OK/failed) för själva inlämningen av mailet, vilket MK kan tolka som att meddelandet ankom gatewayn. Detta protokoll valdes här för att utnyttja Hubs inbyggda förmåga att skicka e-post (enkelt att integrera) och för att isolera MK från SDK-specifika detaljer.

- **SDK REST API (Meddelandeklient–Meddelandetjänst, gränssnitt D):** Detta är det API som mailgatewayn anropar för att överlämna meddelandet till meddelandetjänsten (och som även MK skulle kunna anropa direkt om den vore integrerad). Enligt SDK-specifikationen är detta ett RESTful API baserat på HTTP/HTTPS med JSON-format, definierat av Digg i en OpenAPI-specifikation. API:et omfattar funktioner som *Skicka meddelande*, *Hämta meddelande* och *Radera meddelande*, vilka meddelandetjänsten tillhandahåller till betrodda klienter (som MK-appar eller integrationer). I vårt fall använder mailgatewayn "Skicka meddelande"-operationen. Det skulle kunna se ut som en HTTP POST till t.ex. https://mt.orga.internal/sdkapi/messages med en JSON-body innehållande recipient, sender, subject, body (ev. i krypterad form eller ren text som MT sedan krypterar) och bilagor (kanske Base64-kodade eller via multipart request). Meddelandetjänsten svarar med en bekräftelse om meddelandet accepterats, eventuellt med ett internt ID. All kommunikation över detta API är innanför organisationens säkerhetsgräns, men ändå skyddad (TLS och autentiseringstoken) för att förhindra obehörig åtkomst. **Gränssnitt D** möjliggör dessutom att andra system (t.ex. verksamhetssystem som vill skicka SDK-meddelanden direkt) kan integreras på ett standardiserat sätt utan att behöva känna till AP/AS4-detaljerna.

- **AS4 (OASIS ebMS3.0 AS4):** Detta är **transportprotokollet** som används mellan accesspunkterna över internet (gränssnitt A). AS4 kan ses som ett specialiserat meddelandeprotokoll ovanpå webbtjänster (SOAP) som hanterar säkra, tillförlitliga leveranser av affärsmeddelanden. Inom SDK har man profilerat AS4 enligt CEF eDeliverys rekommendationer, så att alla AP följer samma regler. Några nyckelfakta om AS4 i detta sammanhang:

    - **Web Service-baserat:** AS4-meddelanden skickas som SOAP-meddelanden (XML) över HTTPS. Varje meddelande har en header med metadata (bl.a. unika ID, avsändare, mottagare, dokumenttyp etc) och en body som innehåller själva nyttolasten (ofta i form av en bifogad payloaddel, t.ex. XML eller PDF). Protokollet är payload-agnostiskt – det kan bära godtyckligt binärt eller textbaserat innehåll.

    - **Säkerhet:** AS4 föreskriver användning av WS-Security för signatur och kryptering. Avsändande AP signerar meddelandet digitalt (så mottagaren kan verifiera vem som skickade och att det inte ändrats) och krypterar det för mottagande AP. Dessutom körs det över TLS för transportskydd. Kombinationen ger två lager av kryptering (transport + meddelande) och end-to-end integritet mellan AP:erna.

- **Kvittenser och fel:** AS4 stöder s.k. *reliable messaging*, vilket innebär att mottagaren svarar med en *receipt* (acknowledgement) när ett meddelande mottagits korrekt. I AS4-termer är detta en **AS4 Receipt** som signeras av mottagaren. I SDK-profilen används både denna tekniska kvittens och dessutom applikationskvittenser (meddelandekvittenser som genereras av MT, beskrivna ovan). Om något går fel (t.ex. schemafel, säkerhetsfel) kan en *SOAP Fault* eller negativ kvittens skickas tillbaka. AP:erna hanterar återförsök om kvittens uteblir.

- **One-way vs Two-way:** AS4 kan användas i både enriktade flöden och tvåstegsförlopp. SDK använder främst enriktad leverans med separat kvittens. Det finns även en möjlighet för *AS4 Pull*-mekanism (där mottagande AP själv hämtar meddelanden vid behov), men enligt SDK:s design har man valt push-varianten för normal meddelandeutväxling (pull utvärderades men ansågs mindre fördelaktigt).

- **Interoperabilitet:** Genom att alla accesspunkter är **konformanta** mot AS4-profilen uppnås interoperabilitet. Det betyder att t.ex. en AP med programvara X kan skicka till en AP med programvara Y utan problem. Digg tillhandahåller testverktyg och certifiering för att AP-operatörer ska säkerställa att de följer profilen. Protokollet i sig är standardiserat, så man kan även byta AP-programvara utan att behöva ändra de övriga lagren i systemet (MT eller MK behöver inte veta om bytet sker, så länge gränssnitten bibehålls).

- **Övriga gränssnitt:**

  - *Gränssnitt B* – SDK Adressbok och relaterade API:er: Meddelandetjänsten interagerar med dessa vid behov. Adressbokens Sök-API är ett REST/HTTP-baserat gränssnitt där MT kan söka fram mottagare baserat på t.ex. organisationsnummer, funktionsnamn eller sökord. Även kodverk kan hämtas via ett API för att förstå adressbokens klassificeringar. Dessa API:er är till för system-till-system-uppslag och är säkrade (ofta krävs autentisering och rätt behörighet för att fråga adressboken).

  - *Gränssnitt C* – AP till MT: Som nämnt finns inget enhetligt protokoll här, men vanliga tekniker är JMS, lokala webbtjänster eller REST. I vårt scenario kan man tänka sig att AP har en intern REST-endpoint (https://ap.orga.internal/incoming) som MT pollar, eller att AP skriver till en databas/kö som MT läser ifrån. Oavsett används säkra interna protokoll, ofta ej exponerade utanför organisationens nät. Vi betonar att detta *inte* är en öppen federationsgränsyta, utan en intern integration som kan vara vendor-specifik.

  - *Användargränssnitt:* Slutligen kan nämnas att Hubs som plattform använder webbprotokoll (HTTPS) för att presentera gränssnittet för användaren. Detta är givetvis skyddat med TLS och sessionhantering. Även om det inte är ett "SDK-protokoll" i sig, är det en del av helheten att användaren kommunicerar med servern över HTTP(S) när hen använder meddelandeklienten.

6. Säkerhetsmodell (certifikat, autentisering, loggning)

Säkerhet är centralt i hela SDK-lösningen, då den avser transport av känslig information. Här beskrivs hur lösningen hanterar **certifikat**, **autentisering** och **loggning/spårbarhet** för att upprätthålla en hög säkerhetsnivå.

Certifikat och nyckelhantering: SDK-federationen bygger på en **PKI-baserad tillitsmodell** med flera olika certifikat som används i olika led.

- På **transportnivå** (mellan accesspunkter) används *AP-certifikat* som utfärdas av en betrodd rot (CA) som federationens driftorganisation hanterar. Varje godkänd accesspunktoperatör får certifikat för sina accesspunkter. Dessa certifikat används för att etablera **TLS**-förbindelser (mutual TLS kan krävas, d.v.s. båda AP:erna autentiserar varandra med certifikat) samt för att signera AS4-meddelanden på transportnivå. Endast AP:er med giltiga certifikat (som inte är återkallade) betraktas som betrodda noder i nätverket. Certifikaten hanteras noggrant – det finns rutiner för nyckelbyte och revokation via spärrlistor (CRL/OCSP) som AP:erna ska kontrollera.

- På **meddelandenivå** (end-to-end mellan organisationerna) används *organisationscertifikat* för kryptering och signering av själva meddelandeinnehållet (den s.k. O2O-krypteringen). Varje organisation har ett eller flera certifikat/par nycklar som de använder för att identifiera sig som avsändare och för att kunna dekryptera mottagna meddelanden. Publika nycklar (cert) för detta ändamål delas via en **Certifikatpubliceringstjänst (CertPub)** eller i SMP så att andra kan hämta dem. I vårt flöde innebar det att Org B:s publika nyckel användes av Org A för kryptering, och Org A:s publika nyckel användes av Org B för signaturverifiering. Dessa certifikat är typiskt utfärdade av en eller flera betrodda CA:er inom federationen (skilda från transport-CA:n) och kan vara specifika för t.ex. funktionsbrevlådor. De kan även kallas *organisationscertifikat för meddelandekryptering/signering*. Hantering av dem inkluderar att nya certifikat måste spridas i nätverket i god tid och att utgångsdatum bevakas.

- **Användarcertifikat** är inte explicit en del av SDK-arkitekturen, men om den lokala implementationen (t.ex. Hubs) skulle använda klientcertifikat eller smartkort för användarinloggning, kunde det förekomma. Dock är det vanligare med Federationens *identitetsfederation* som syftar mer på organisationers identitet än enskilda användare. Användarna autentiseras normalt med andra metoder (se nedan).

Sammanfattat garanterar certifikatmodellen att både transportkanalen och meddelandet självt är krypterat och signerad i två lager, vilket uppfyller höga krav på konfidentialitet, integritet och spårbarhet. Alla certifikat har tillhörande nycklar som måste skyddas väl i driftmiljön (t.ex. lagras i HSM eller motsvarande).

Autentisering och behörighet: Autentisering sker på flera nivåer:

- **Användarautentisering:** Användare som ska skicka/läsa meddelanden loggar in i Hubs. Hubs stödjer integration med katalogtjänster (LDAP/AD), SSO (SAML/OAuth) eller t.ex. BankID. I en offentlig sektorskontext kan hög tillitsnivå

krävas för åtkomst till känsliga meddelanden – BankID eller SITHS-kort kan användas för stark autentisering av användare. När användaren är inloggad och öppnar MK-appen, kontrolleras även att hen har behörighet att använda den (kanske endast vissa roller får skicka säkra meddelanden). Behörighetsstyrning kan även ske per funktionsbrevlåda: t.ex. bara vissa handläggare ser viss brevlåda. Hubs hanterar detta via sina grupp/rättighetsinställningar.

- **Systemautentisering (intern):** Mailgatewayn autentiserar MK:s SMTP-anslutning. Det kan ske antingen via IP-filter (t.ex. bara Hubs-serverns IP är tillåten), eller via SMTP AUTH med ett tekniskt konto. Detta förhindrar att obehöriga interna system försöker posta in i mailgatewayn. Vidare autentiserar mailgatewayn sig mot meddelandetjänstens API, vanligtvis genom en API-nyckel eller token. Meddelandetjänsten har konfigurerade behöriga klienter – i detta fall borde gatewayn vara registrerad. Kanske används OAuth2 med klientuppgifter eller helt enkelt en delad hemlighet. Detta skyddar mot att någon annan på internnätet skulle kunna skicka falska meddelanden in i MT.

- **Federationsautentisering (extern):** Accesspunkterna autentiserar varandra vid varje kommunikation. Här används de ovan nämnda AP-certifikaten inom TLS. När AP A kontaktar AP B över TLS, presenteras A:s certifikat som B verifierar mot den betrodda rotkedjan (och vice versa om mutual TLS). Endast om certifikaten är giltiga och icke-spärrade fortsätter utbytet. På så sätt vet AP B att det är en godkänd avsändare. Innehållet i meddelandet är dessutom signerat av Org A:s organisationscert, vilket AP B och MT B kan använda för att veta exakt vilken organisation som är avsändare (det finns en koppling mellan Org A och dess certifikat i adressboken). SDK:s **identitetsfederation** innebär just att det finns regler och tekniska metoder för att säkerställa parternas identitet – både teknisk (AP) och organisatorisk (meddelandes avsändare)meaplus.com.

- **Åtkomstkontroll:** Utöver autentisering finns behörighetsstyrning: meddelandetjänsten ser till att en användare eller ett system bara får göra det den är tillåten till. T.ex. MK-appen får via API bara skicka från de funktionsbrevlådor användaren har rätt till. Mottagande MT kontrollerar att inkommande meddelande är adresserat till en giltig mottagare hos sig; annars kan det avvisas. Hubs MK-appen kan implementera ytterligare kontroller, t.ex. att användaren inte skickar hemlig info till en mottagare som inte är godkänd (det kan finnas policykontroller på klientnivå också).

Loggning och spårbarhet: För att uppfylla krav på spårbarhet (t.ex. vid sekretesshantering och arkivering) loggas händelser i samtliga delar av systemet:

- **Meddelandeklientloggar:** MK (Hubs) kan logga användaråtgärder som "Användare X skapade/skickade meddelande Y till mottagare Z vid tidpunkt T" och "Användare X läste meddelande W vid tidpunkt T2". Dessa loggar kan lagras i Hubs loggsystem och eventuellt exporteras till en central loggtjänst. De är viktiga för att kunna visa t.ex. när ett visst meddelande skickades eller om/ när en mottagare läste det (om läskvittenser används).

- **Mailgateway loggar:** Gatewayn loggar alla inkommande SMTP-transaktioner och utgående API-anrop. Detta inkluderar tidsstämplar, avsändare/mottagare och om anropet till MT lyckades eller eventuella fel. Vid felsökning kan dessa loggar brukas för att se om meddelandet kom in i systemet.

- **Meddelandetjänst loggar:** MT för noggrann logg över varje meddelande: när det mottogs (från MK/gateway), meddelandets ID, till vilken mottagar-Org det skulle, när det krypterades, när det skickades till AP, eventuella valideringssteg samt mottagna kvittenser. För inkommande meddelanden loggas motsvarande: från vem, när det hämtades från AP, resultat av signaturkontroll, etc. MT kan även logga innehållsmetadata (men oftast inte klartext innehåll, för att skydda sekretessen – krypterat innehåll kan dock loggas i säkert förvar om arkivering krävs). MT-loggar är centrala för att spåra meddelandeflödet end-to-end inom organisationen. De används vid incidenthantering (om någon påstår att ett meddelande inte kom fram, kan loggarna visa alla steg).

- **Accesspunkt loggar:** AP loggar transporthändelser: utgående skickade meddelanden (med MessageID, mottagare, storlek, tidpunkt), mottagna meddelanden, skickade och mottagna transportkvittenser, fel osv. Dessa loggar är ofta detaljerade och kan inkludera protokollspecifika ids. De är viktiga för teknisk spårbarhet och för att bevisa leveranser (AP-loggen kan visa att AP B returnerade en ACK kl 10:31 osv). Även säkerhetshändelser loggas, t.ex. misslyckade försök att ansluta med fel certifikat.

- **Revisionsloggar:** Utöver driftloggar kan systemet generera högre nivåns **revisionsposter** för varje meddelande som skickas, som sammanfattar hela resan (t.ex. "Meddelande X skickat av OrgA, mottaget av OrgB, kvitterat kl Y"). Dessa kan användas för att mata in i ett ärendehanteringssystem eller arkiveras för framtida revision.

Sammantaget utgör ovanstående mekanismer en **gedigen säkerhetsmodell**: certifikat säkerställer identiteter och kryptering; autentisering/behörighet skyddar åtkomst på alla nivåer; loggning och kvittenser ger full spårbarhet och möjlighet att upptäcka och utreda eventuella incidenter. Lösningen följer principen om *defense in depth* (flerskikts-säkerhet) och uppfyller de krav som ställs i SDK:s regelverk för transport av känslig information.

7. Drift och redundans (klustring, återställning, integrationer)

Vid införande av en SDK-lösning i produktion är det viktigt att beakta hur systemet ska drivas robust och kunna återhämta sig från fel. Nedan diskuteras klustring (hög tillgänglighet), katastrofåterställning samt integrationer med omgivande infrastruktur.

Klustring och hög tillgänglighet: För att undvika single points of failure kan flera av komponenterna köras i kluster eller redundanta konfigurationer:

- *Hubs MK:* Hubs stödjer drift i kluster (flertal applikationsservrar bakom en lastbalanserare, gemensam databas och filsystem). Genom att köra Hubs i HA-modus kan meddelandeklienten fortsätta vara tillgänglig även om en server faller bort. Detta är viktigt om många användare är beroende av tjänsten. Lastbalansering säkerställer också bättre prestanda vid hög trafik.

- *Mailgateway:* En mailgateway kan sättas upp med primär/sekundär MX-poster (mail exchangers) – alltså två instanser av SMTP-servern där om en är nere kan MK försöka nästa. Alternativt kan gatewayn vara ett script på Hubs-servern

själv, men då delar de öde. I en fristående variant kan två gateway-instanser köra parallellt och ha gemensam koppling till MT (t.ex. båda anropar samma MT endpoint). Det är dock viktigt att undvika dubbelsändningar; normalt löser man detta med att endast en gateway är aktiv åt gången, eller att Hubs har fallback-server vid misslyckad leverans.

- *Meddelandetjänst:* MT bör absolut vara redundant. Ofta implementeras MT som en stateless applikation som körs på flera noder mot en gemensam databas/kö. Man kan ha en aktiv-aktiv klustring där inkommande API-anrop (från MK/gateway) och AP-leveranser fördelas mellan noder. Köhanteringen (t.ex. JMS-servrar eller databas) kan i sig klustras eller ha failover. Om en MT-instans skulle krascha mitt i en sändning, så ska nästa instans kunna ta över meddelandet från kön och skicka det. Sessionstillstånd för kryptering etc. är minimalt eftersom varje meddelande behandlas isolerat. En utmaning är hantering av kvittenser och undvika att två noder behandlar samma kvittens; men med rätt design (t.ex. unika konsumenter på kön) går det bra. Ett kluster av MT-noder förbättrar också prestanda genom att parallellisera hanteringen av flera meddelanden.

- *Accesspunkt:* Accesspunkten kan klustras eller köras med failover. Eftersom AP är kopplad till DNS-adresser (SMP pekar på en adress), bör en failover ha samma adress eller DNS som uppdateras snabbt. AP:n bör också ha redundans i nätverksväg (flera uplinks etc) eftersom den kommunicerar utåt. Geografisk redundans (två datacenter) kan användas för AP om meddelandeflödet är mycket kritiskt, vilket ger motståndskraft mot t.ex. strömavbrott eller internetstörningar i en region. SDK-federationen har krav att AP-operatören upprätthåller hög tillgänglighet för att inte bli en flaskhals för andra.

- *Gemensamma tjänster:* Även om de inte driftas av oss direkt, är det värt att notera att de centrala tjänsterna (adressbok, SMP, Certifikat-tjänst) hos Digg/Inera också är redundanta på deras sida för att alltid vara nåbara. Vår MT cachar dessutom troligen adressdata så att ett kort avbrott där inte stoppar skickande.

Återställning och katastrofberedskap: Trots redundans kan allvarliga incidenter inträffa, därför behövs planer för backup och återställning:

- *Backup av data:* Hubs databas och lagring (innehållande meddelanden, bilagor, loggar) säkerhetskopieras regelbundet. Detsamma gäller meddelandetjänstens databas/köer. Rullande backuper mer 10 versioner finns alltid, med möjlighet att mer frekvent exportera meddelandemetadata.

- *Katastrofscenario:* I händelse av en större katastrof (t.ex. att hela datacentret förloras) ska en *Disaster Recovery*-lösning finnas. ITSL har parallell driftmiljö (backup-site) dit driften automatiskt skiftas. Eftersom federationen litar på DNS och certifikat, kan en organisation ha två registrerade accesspunktsadresser (primär och sekundär) eller snabbt ändra DNS för AP:ns adress till den nya platsen. Under tiden bör meddelanden i nätverket köas hos avsändarna (tack vare asynkron modell). Organisationen måste kommunicera eventuella längre avbrott till federationsoperatören om det påverkar trafiken.

- *Testade rutiner:* För att vara förberedda testas backupåterläsning och failover-scenarion periodiskt. Exempel på tester är: stäng ned en MT-nod och se att

andra tar över; återläs en databas till testmiljö och se att index och data är intakta; prova att swappa AP till sekundär. Detta ger förtroende att redundansen faktiskt fungerar när det gäller.

Integrationer med övrig infrastruktur: Lösningen behöver passa in i organisationens IT-miljö och processer:

- *Användarkatalog och SSO:* Hubs MK integreras sannolikt med befintlig användarkatalog (t.ex. Active Directory) för enkel hantering av konton och single sign-on. Detta underlättar administration eftersom användarna kan använda sina vanliga inloggningsuppgifter.

- *Klientintegrering:* För slutanvändare kan det vara värdefullt att meddelandeklienten integrerar med deras arbetsflöde. Exempel: om organisationen har ett ärendehanteringssystem kan man integrera så att ett säkert meddelande kan kopplas till ett ärende-ID eller lagras i diariesystemet för arkivering. Hubs API:er eller webhooks kan användas för att, när ett meddelande kommer in, trigga en överföring av PDF:er eller info till ett journalsystem. Sådana integrationer är specifika för verksamheten men viktiga att planera in.

- *Övervakning och larm:* Alla komponenter bör integreras med organisationens övervakningssystem (t.ex. Nagios, Zabbix, Splunk). Det ska finnas larm för om någon komponent går ner (Hubs, MT, AP, etc.), om köer växer onormalt, om meddelanden fastnar eller kvittenser uteblir. Speciellt meddelandetjänsten kan exponera statistik – antal skickade, fel, etc – som kan övervakas. Loggar kan skickas till en SIEM-plattform för att upptäcka säkerhetsincidenter (t.ex. obehöriga inloggningsförsök).

- *Uppdateringar och kompatibilitet:* SDK-standarderna utvecklas, så systemet bör kunna uppdateras med nya versioner (t.ex. ny AS4-profil, nya kodverk i adressboken etc.). Integration med leverantörernas roadmap är en aspekt. Detta innebär att arkitekturen med separata lager är bra: man kan byta ut/uppgradera MK, MT eller AP var för sig så länge de fortsätter följa standardgränssnitten

- *Prestanda och skalning:* Integrationsmässigt måste man också planera för volym. Om plötsligt mängden meddelanden ökar (fler användare, fler anslutna organisationer) ska systemet skala. Här hjälper klustring (som nämnts). Även Hubs integration med systemresurser (CPU, minne, I/O) bör optimeras så att stora bilagor inte tynger ner systemet. AP:n och MT:n kan behöva justeras (t.ex. fler trådar, mer minne) för att hantera toppar. Att integrera med en snabb databas och eventuellt cacheservrar (Redis etc.) kan vara aktuellt i stor drift.