

# Java-basierte WS-Security Implementierung

Detlef Sturm  
Senior Manager  
Product Technology  
Beta Systems Software AG

Bitkom  
SOA & Security

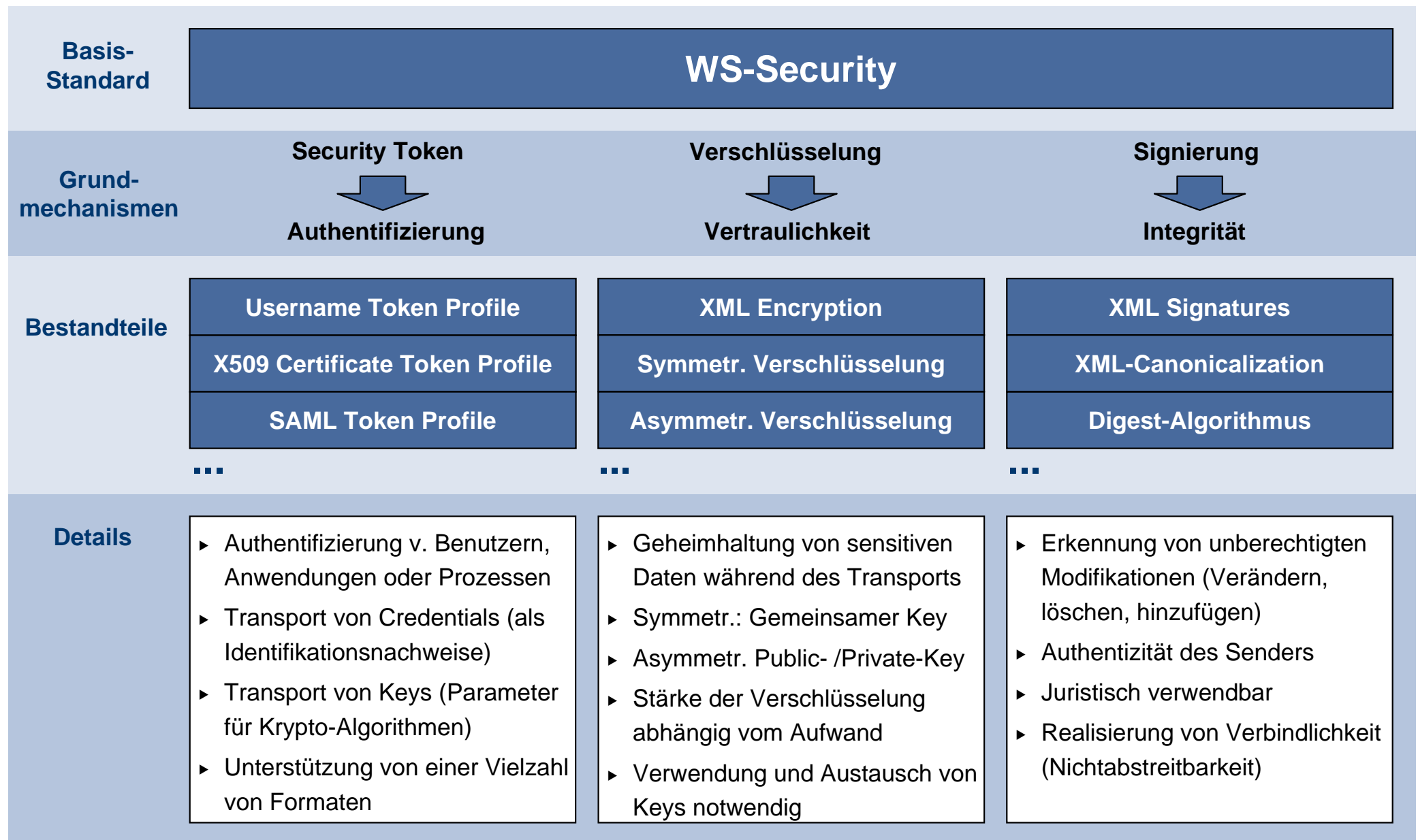
**Frankfurt, 12.03.2008**



## Agenda (Teil 1)

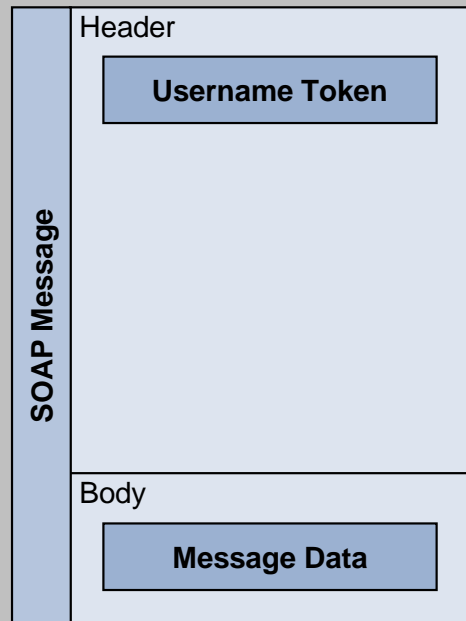
- ▶ WS-Security: Grundmechanismen und Bestandteile (Überblick)
- ▶ Zusammenspiel der Mechanismen und Aufgaben von Signaturen
- ▶ Grundsätzliche Design-Prinzipien für Security-Implementierungen
- ▶ Architektur-Patterns für Security-Implementierungen
- ▶ Granularität und Interoperabilität von Security





( Timestamp ist ein weiterer Mechanismus im Rahmen von WS-Security: Verhinderung von Replay-Attacken)

## Message-Level Security



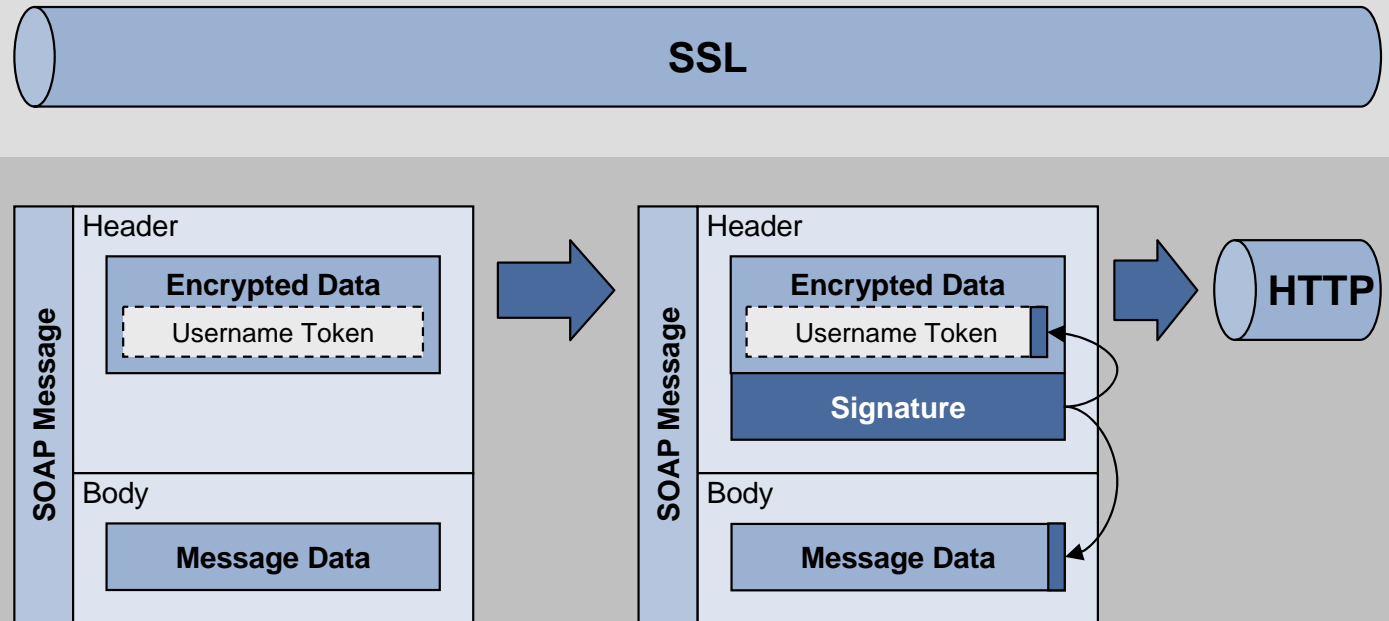
### Authentifizierungsdaten

- ▶ **Username-Token** als Container für Auth.-Daten
  - ▶ User-ID, Passwort

#### Problem:

- ▶ Daten sind lesbar

## Transport-Level Security



### Vertraulichkeit

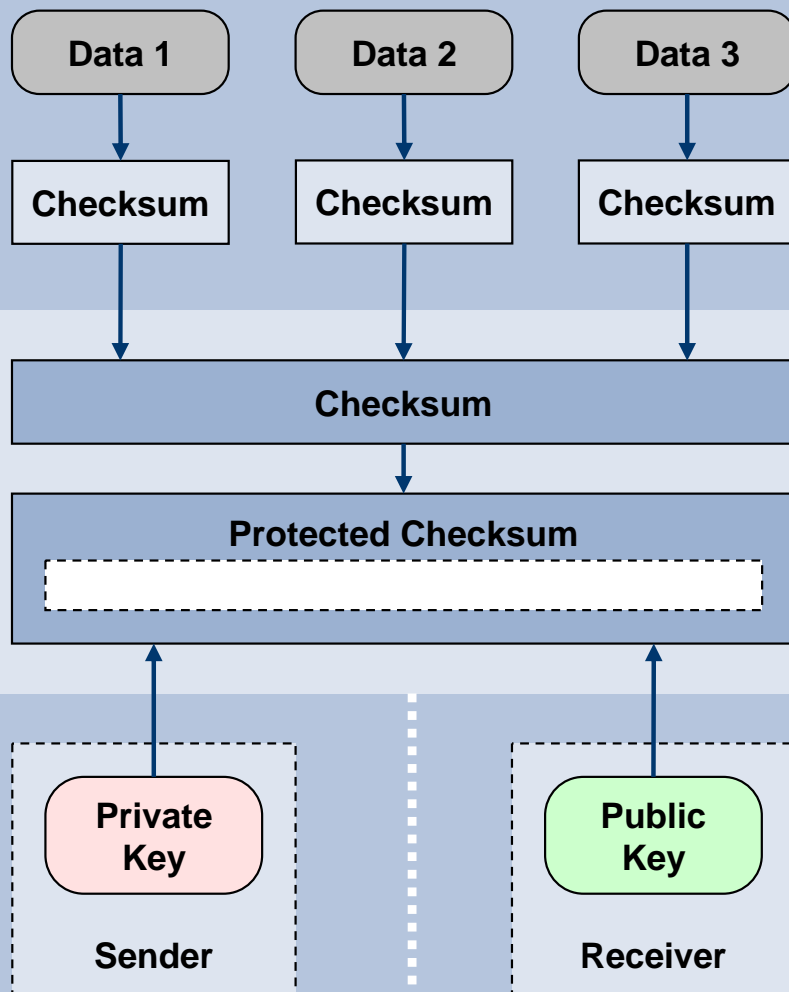
- ▶ **Verschlüsselung** der Daten via WS-Security
  - ▶ Alternativ: SSL

#### Problem:

- ▶ Username-Token ist nicht gekoppelt an die Nachricht

### Integrität

- ▶ **Signatur** u.a. als Klammer für mehrere Datenblöcke
- ▶ Nachweis für die Authentizität der Nachricht



## Datenintegrität

- ▶ Erkennung von Datenmanipulationen (Verändern, Löschen, ...)
- ▶ Maßnahme: Bildung einer Prüfsumme (Hash, Digest)
- ▶ Algorithmus: SHA1

## Nachrichtenauthentizität

- ▶ Schutz vor Daten- und Prüfsummenmanipulationen
- ▶ Verknüpfung von mehreren Datenblöcken
- ▶ Maßnahme: Gemeinsame Prüfsumme und Verschlüsselung
- ▶ Algorithmen: SHA1, HMAC (symmetr. Verschlüsselung)

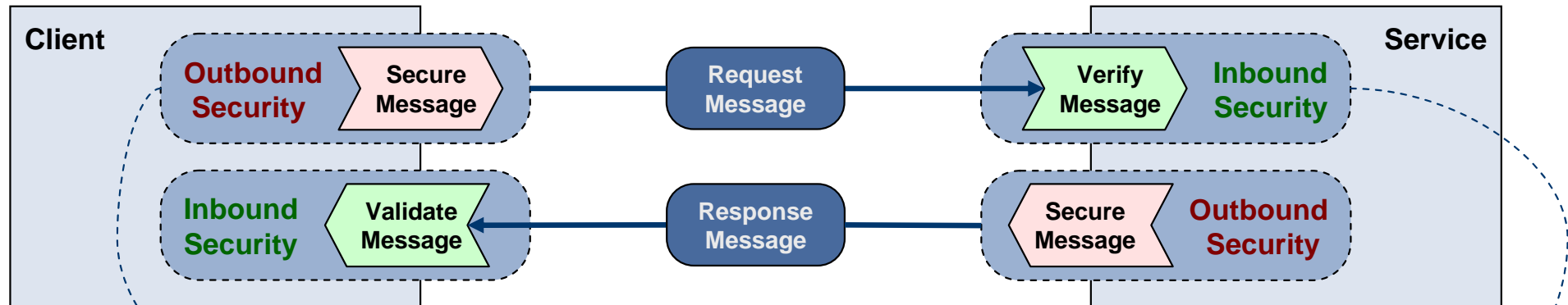
## Senderauthentizität

- ▶ Die Verwendung einer Private-Keys bietet die Möglichkeit einer Senderauthentizität zu ermitteln
- ▶ Maßnahme: Asymmetrische (Public-Key) Verschlüsselung
- ▶ Algorithmen: DSA, RSA

## Dilemma

- ▶ Signaturen haben als Ergebnis nur: **Richtig oder Falsch** (Keine Details bei nicht Übereinstimmung)
- ▶ **Interoperabilität** ist die Herausforderung: Normalisierung, Digestbildung, Verschlüsselung, Abbildung

# Outbound vs. Inbound Security: Prinzip und Aktivitäten

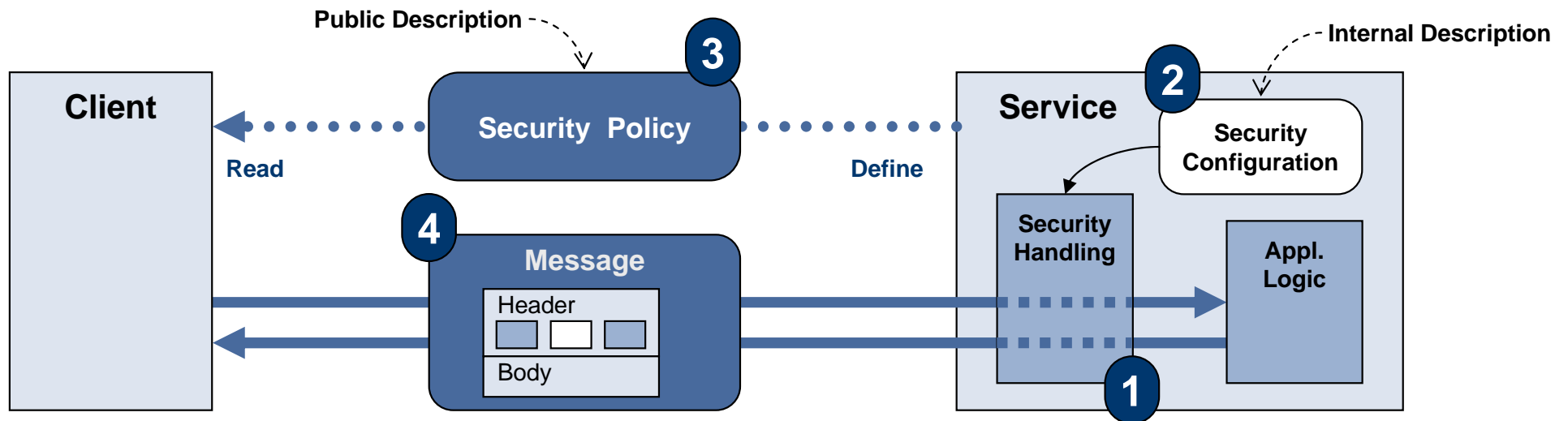


## Outbound-Aktivitäten (Client)

## Inbound-Aktivitäten (Service)

		Überprüfung hinsichtlich der Einhaltung der Security-Policy	
<b>Timestamp</b>	<ul style="list-style-type: none"> <li>▶ Bildung einer <b>Message-ID</b> (Zufallswert)</li> <li>▶ Angabe einer <b>Gültigkeitszeitraum</b>: Erstellungs- und Verfallszeit</li> </ul>	▶	<ul style="list-style-type: none"> <li>▶ <b>Prüfung</b> hinsichtlich der Einmaligkeit</li> <li>▶ <b>Speicherung</b> und Verwaltung der Message-IDs</li> </ul>
<b>Security-Token</b>	<ul style="list-style-type: none"> <li>▶ Abfrage User-ID und Passwort (Logon)</li> <li>▶ <b>Einbetten der Daten</b> in den SOAP-Header</li> </ul>	▶	<ul style="list-style-type: none"> <li>▶ <b>Authentifizierung</b>: Überprüfung von User-ID und Passwort</li> <li>▶ <b>Autorisierung, Auditing, Accounting</b></li> </ul>
<b>Verschlüsselung</b>	<ul style="list-style-type: none"> <li>▶ <b>Verschlüsselung</b> der Datenblöcke nach spez. Algorithmus, dazu Key-Zugriff</li> <li>▶ Key-Infos in den Header</li> </ul>	▶	<ul style="list-style-type: none"> <li>▶ <b>Entschlüsselung</b> der Datenblöcke nach spez. Algorithmus, dazu Zugriff auf dazugehörigen Key</li> </ul>
<b>Signaturen</b>	<ul style="list-style-type: none"> <li>▶ Pro Block: Normalisierung, Digestbildung</li> <li>▶ <b>Signierung</b> nach spez. Algorithmus</li> <li>▶ Key-Zugriff und Key-Infos in den Header</li> </ul>	▶	<ul style="list-style-type: none"> <li>▶ Pro Block: Normalisierung, Digestbildung</li> <li>▶ <b>Entschlüsselung</b> der Signatur und <b>Vergleich</b> der Digests</li> </ul>

# Grundsätzliche Design-Prinzipien für Security-Implementierungen



1

## Self-contained-Security

- ▶ Trennung Security-Handling und Appl-Logik
- ▶ Unterschiedliche Lifecycles
- ▶ Pluggable Security
- ▶ Anbindung von verschiedenen Frameworks

2

## Security-by-Configuration

- ▶ Einsatzspezifische Anforderungen
- ▶ Anpassbare Security
- ▶ Keine hard-coded Security
- ▶ Interne Beschreibung
- ▶ Einstellungen für das interne Handling

3

## Policy-driven-Security

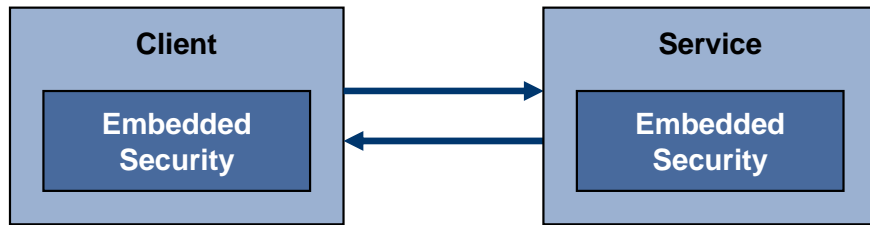
- ▶ Vertrag zwischen Client und Service
- ▶ Beschreibung der Anforderungen (**MUSS**)
- ▶ WS-SecurityPolicy als Basis-Standard
- ▶ Einbettung in der WSDL
- ▶ Öffentliche Beschreibung

4

## Self-describing-Security

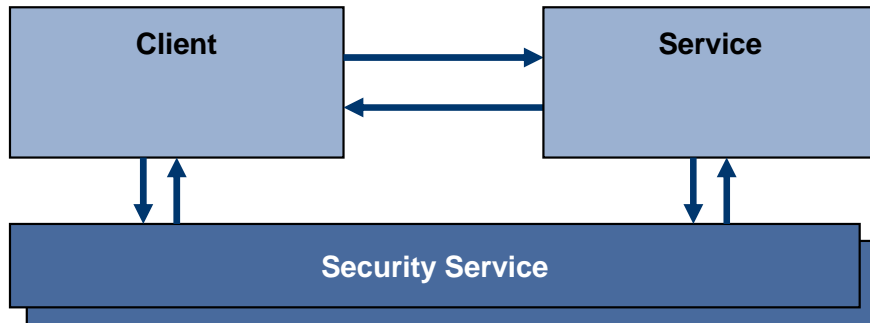
- ▶ Beschreibung der verwendeten Security-Mechanismen
- ▶ Keine Abstimmungen aller Details zwischen Client und Service (**KANN**-Eigenschaften)
- ▶ Grundlage: WS-Security

( Prinzipien gelten auch für die Client-seitige Realisierung der Security)



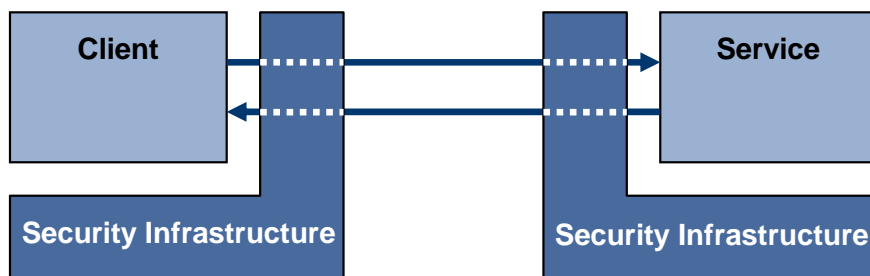
## Embedded Security

- ▶ Unabhängig, aber komplex, proprietär, monolythisch
- ▶ Keine zentrale Administrationsmöglichkeit
- ▶ Keine Trennung von Security-Handling und Applikationslogik



## Security as a Service

- ▶ Bereitstellung der Security-Funktionalität als Dienst
  - ▶ Wiederverwendung, zentrales Management
- ▶ Konsument und Anbieter benutzen aktiv den Security-Service
- ▶ Mehrere Services für unterschiedliche Aufgaben

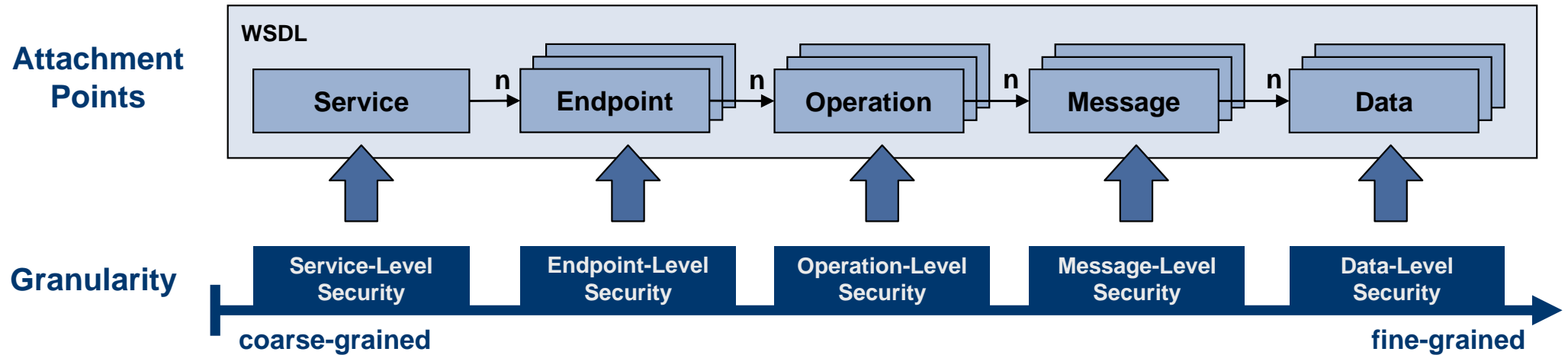


## Security as Infrastructure

- ▶ Entkopplung der Applikationslogik vom Security-Handling
- ▶ Security-Handling ist für Konsument und Anbieter transparent
- ▶ Realisierung als Software oder auch Hardware

Oder Mischformen: Client und Service verwenden unterschiedliche Pattern.

## Granularität abhängig von der Zuordnung



- ▶ Security-Granularität wird bestimmt durch die Zuordnung zu den verschiedenen Web Service Elementen  
Arten:
  - ▶ **Service-Level Security:** gilt für den kompletten Service, unabhängig von Ports und Operationen
  - ▶ **Endpoint-Level Security:** gilt nur die jeweiligen Port (z.B. HTTP-Port 8081)
  - ▶ **Operation-Level Security:** gilt nur für die jeweilige Operation
  - ▶ **Message-Level Security:** gilt für nur für die jeweilige Nachricht (z.B. Request oder Response)
  - ▶ **Data-Level Security:** gilt nur für bestimmte Datenblöcke innerhalb einer Nachricht (z.B. Kreditkarten-Nr.)
- ▶ WS-SecurityPolicy definiert die Granularität in Form von verschiedenen Policy Subjects
  - ▶ Separater Standard: WS-PolicyAttachment

## Grundlegende Interoperabilität (Basics)

- ▶ Konformität hinsichtlich der Basisspezifikationen
- ▶ Welche Informationen sind für die jeweiligen Algorithmen notwendig?
- ▶ Welche Struktur und Reihenfolge von Elemente sind zu beachten?
- ▶ Wie sind die jeweiligen Security-Token zu verwenden?
- ▶ ...



**WS-I:  
Basic Security Profile**

## Interoperabilität der Algorithmen

- ▶ Standard-konforme Implementierung der einzelnen Algorithmen
- ▶ Algorithmen
  - ▶ Canonicalization (XML-Normalisierung), abhängig vom (XPath-)Parser
  - ▶ Digest-Bildung (SHA1), Verschlüsselungen (MAC, RAS, TripleDES, ...)
  - ▶ Bei Signaturen besonders kritisch → Richtig-oder-Falsch Entscheidung

## Interoperabilität der Profile (Choreografie)

- ▶ Vordefinierte Kombinationen von Security-Token, Verschlüsselung, Signaturen
  - ▶ Reihenfolge, Algorithmen, Arten der Security-Token, Datenblöcke
- ▶ Anerkannte Best Practices (Semi-Standards?), bisher keine Standardisierungen
- ▶ Beschreibung der Anforderungen via WS-SecurityPolicy
  - ▶ z.T. wird die Abfolge der einzelnen Algorithmen definiert



## Agenda (Teil 2)

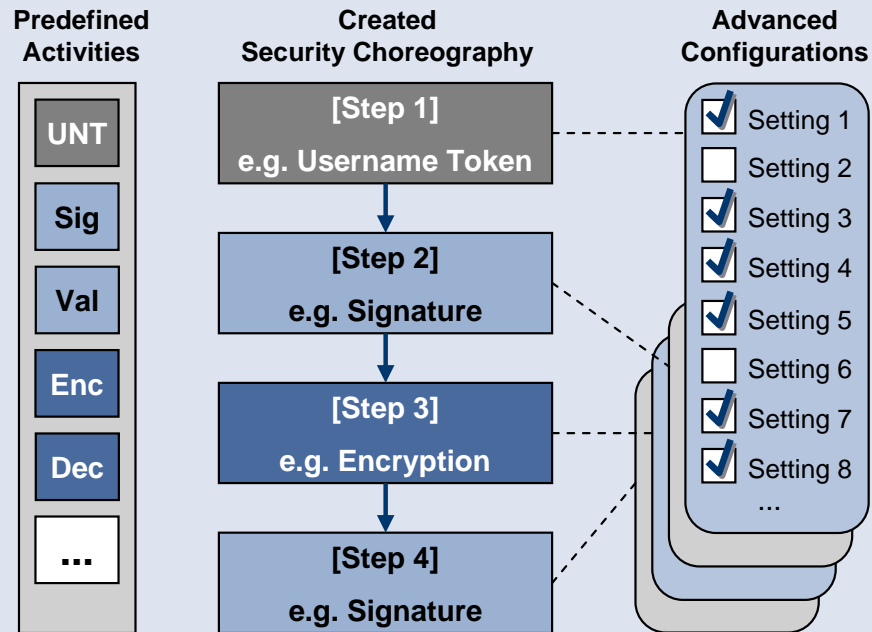
- ▶ Java-basierte Implementierungen von WS-Security (Überblick)
- ▶ Grundlegende Ansätze für Security-by-Configuration
- ▶ Kurze Vorstellung der Implementierungen von Sun



- ▶ Momentan existiert kein Java-Standard für die WS-Security Implementierung
  - ▶ Kein JSR (Java Specification Request) im Rahmen des Standardisierungsprozesses
  - ▶ Relevante Standards: XML-Signature (JSR105) und XML-Encryption (JSR106)
- ▶ Jede Applikations-Plattform (SOAP-Engine) liefert eine eigene WSS-Implementierung
- ▶ Open Source Implementierung
  - ▶ **WSS4J** von Apache (Low-level Ansatz, API-orientiert)
  - ▶ Wird wiederum von Open Source Anwendungen genutzt (z.B. Axis)
- ▶ Sun: Kandidaten für eine Referenzimplementierung
  - ▶ **XWSS 2.0** (XML and Web Service Security)
    - ▶ Wiederverwendbare API in eigenen Handler
    - ▶ **Step-by-Step Konfiguration**
  - ▶ **WSIT** (Web Service Interoperability Technology) (XWSS 3.0)
    - ▶ Im Rahmen des GlassFish-Projektes
    - ▶ Größten Teils versteckt als Container-Funktionalität
    - ▶ **Profile-orientierte Konfiguration**
    - ▶ Unterstützt bereits WS-SecurityPolicy

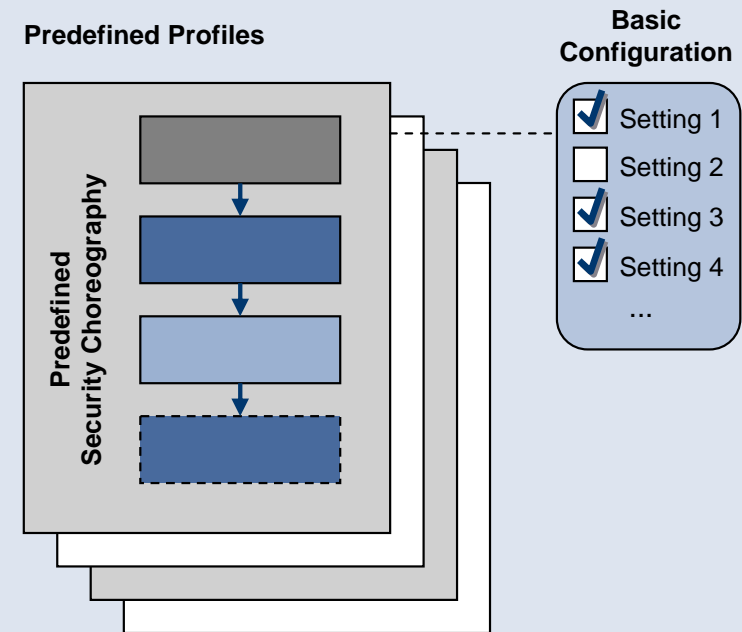


## Step-by-Step Approach



- ▶ Vordefinierte (Einzel-) Aktivitäten
- ▶ Erstellung einer eigenen Security-Choreografie durch Aneinanderreihung von Aktivitäten
- ▶ Jeweils eine Outbound- und Inbound-Choreografie
- ▶ Low-level Konfiguration der einzelnen Aktivitäten
- ▶ Hohes Verständnis über Sec.-Prinzipien notwendig
- ▶ Flexibel hinsichtlich der Abstimmung der Fähigkeiten zwischen Clients und Service

## Profile-oriented Approach



- ▶ Vordefinierte Profile (Templates)
- ▶ Profile haben eine vordefinierte Security-Choreografie, die nur eingeschränkt verändert werden kann
- ▶ Profil gilt für Outbound- und Inbound-Verhalten
- ▶ High-level Konfiguration des Profiles
- ▶ Erfordert („nur“) Grundwissen über Security
- ▶ Interoperabilität als kritischer Faktor
- ▶ Ganzheitlich, bedarf aber einer Standardisierung

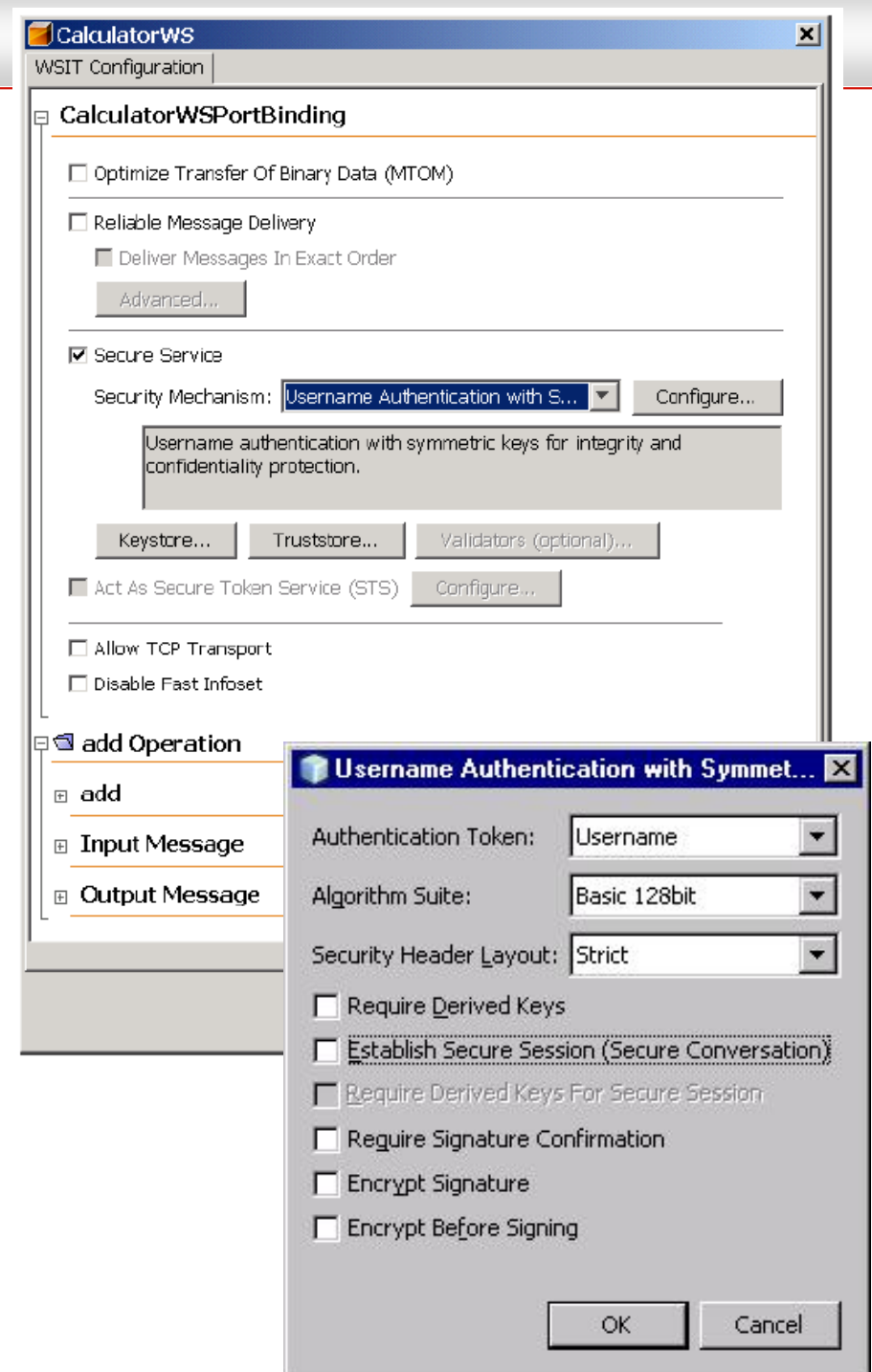
```
<sc:SecurityConfiguration
  dumpMessages="xs:boolean"
  enableDynamicPolicy="xs:boolean">
  Outbound
  <sc:Timestamp ... />
  <sc:UsernameToken ... />
  <sc:SAMLAssertion ... />
  <sc:Sign ... >
  ...
  </sc:Sign>
  <sc:Encrypt ... >
  ...
  </sc:Encrypt>
  Inbound
  <sc:RequireTimestamp ... />
  <sc:RequireUsernameToken ... />
  <sc:RequireSAMLAssertion ... />
  <sc:RequireSignature ... >
  ...
  </sc:RequireSignature>
  <sc:RequireEncryption ... >
  ...
  </sc:RequireEncryption>
  <sc:OptionalTargets ... >
  ...
  </sc:OptionalTargets>
</sc:SecurityConfiguration>
```

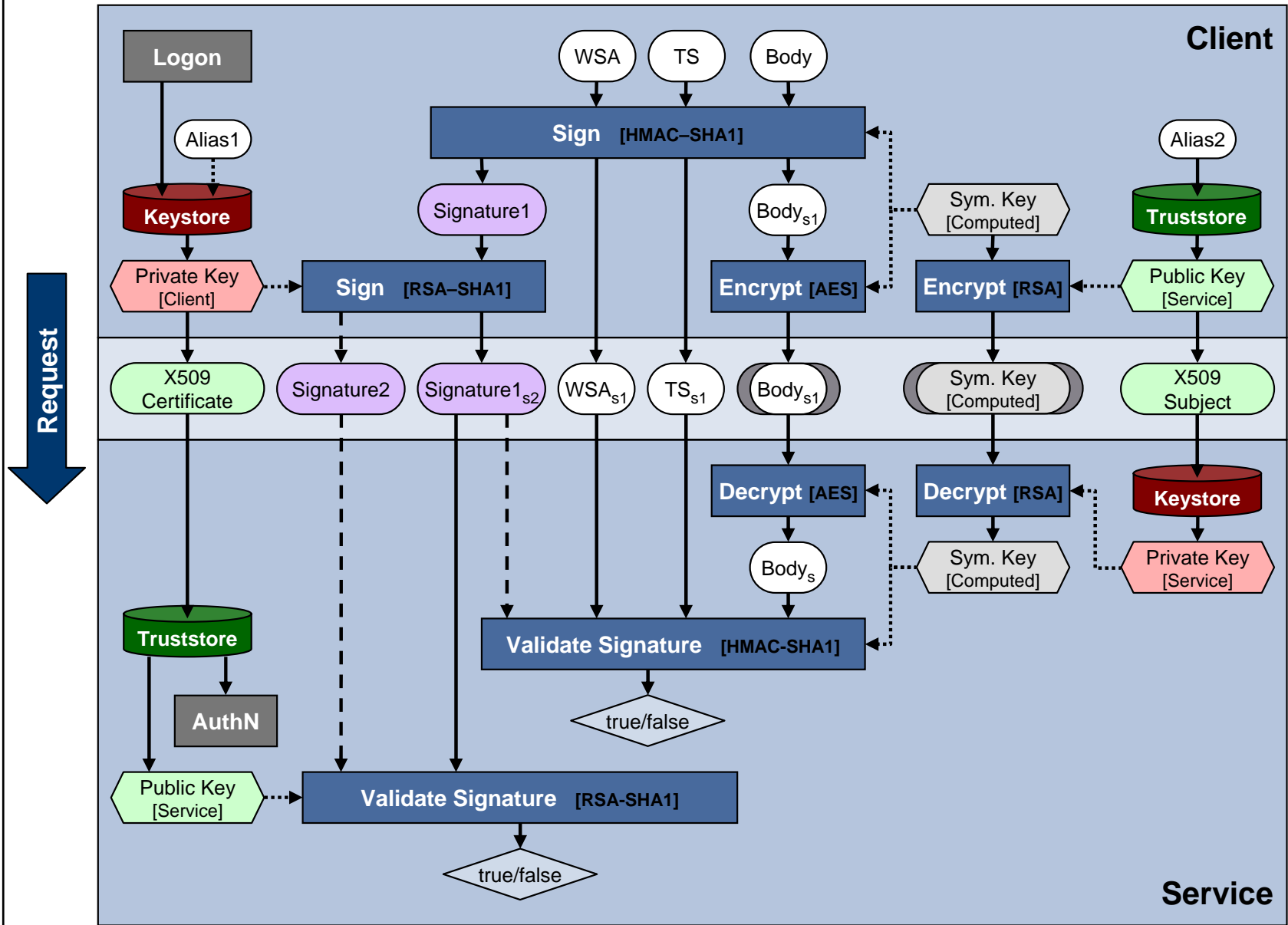
- ▶ Konfiguration nach dem Step-by-Step-Ansatz
  - ▶ Einstellungen werden über XML-Datei getätigt
  - ▶ Keine benutzerfreundlichen Dialoge
  - ▶ Verwendbar für die Service- und Client-seitige Implementierung
- ▶ Outbound- und Inbound-Einstellungen
  - ▶ Inbound-Elemente repräsentieren die Security-Anforderungen, die die eingehende Nachricht erfüllen **muss**
  - ▶ Outbound: Security-Handling für ausgehende Nachrichten
- ▶ Security-Choreografie
  - ▶ Ablauf wird durch die Reihenfolge der Elemente in der XML-Datei bestimmt



Neun verschiedene Profile:

- ▶ **Username Authentication with Symmetric Key**
  - ▶ Gemeinsamer symmetrischer Key
- ▶ **Mutual Certificates Security**
  - ▶ Zertifikatsbasierte Authentifizierung
- ▶ **Transport Security**
  - ▶ SSL
- ▶ **Message Authentication over SSL**
  - ▶ Signierte Nachricht
- ▶ **SAML Authorization over SSL**
  - ▶ SAML Token für Autorisierung
- ▶ **Endorsing Certificate**
  - ▶ Zertifikate zur Bestätigung der signierten Nachricht
- ▶ **SAML Sender Vouches with Certificates**
  - ▶ Zertifikatsbasierte SAML-Token
- ▶ **SAML Holder of Key**
  - ▶ Signierte SAML-Assertion
- ▶ **STS Issued Token**
  - ▶ Token von Token-Service (WS-Trust)





- ▶ Zwei Signaturen
- ▶ Verschiedene Algorithmen
- ▶ Symmetrische Verschlüsselung für Nachrichten-Authentizität
- ▶ Asymmetrische Verschlüsselung für die Bestätigung
- ▶ Verwendbar für Authentifizierung

( WSA: WS-Addressing, TS: Timestamp)

**Vielen Dank für die  
Aufmerksamkeit**

