REST VS SOAP

<https://msdn.microsoft.com/en-US/magazine/Dd942839.aspx>

**Which is better, REST or SOAP?**

Both REST and SOAP are often termed "Web services,"

 REST is an architectural style for building client-server applications. SOAP is a protocol specification for exchanging data between two endpoints.

because REST relies on the semantics of HTTP, requests for data (GET requests) can be cached.

RPC systems generally have no such infrastructure (and even when performing RPC using SOAP over HTTP, SOAP responses can't be cached because SOAP uses the HTTP POST verb, which is considered unsafe).

**What about security? Isn't SOAP more secure than REST?**

secure conversations over HTTP is now called Transport Layer Security (TLS), SSL is still the name most commonly used. What is true is that a SOAP-based service, because of the extra protocols specified in the various WS-\* specifications, does support end-to-end message security. This means that if you pass SOAP messages from endpoint to endpoint to endpoint, over the same or different protocols, the message is secure. If your application needs this particular feature, SOAP plus WS-\* is definitely the way to go.

**What about transactions?**

Here is another area in which SOAP and WS-\* have explicit support for an "advanced" feature and REST has none. WS-Atomic Transactions supports distributed, two-phase commit transactional semantics over SOAP-based services. REST has no support for distributed transactions. You can have a resource called Transaction. When your client needs to do something transactional (such as transferring money between two bank accounts), the client creates a Transaction resource that specifies all the correct resources affected (in my example, the two bank accounts) by doing a POST to the Transaction factory URI. The client can then perform updates by sending a PUT to the transaction URI and close the transaction by sending a DELETE to the URI.

This, of course, requires some amount of hand-coding and explicit control over your system, whereas the WS-Atomic Transactions system is more automatic

If your system absolutely needs atomic transactional semantics across diverse systems, WS-Atomic Transactions is probably the way to go.

**What about interoperability? Isn't SOAP supposed to be about interoperability? Isn't SOAP more interoperable than REST?**

If you define interoperability as the technical ability to communicate between two divergent endpoints, I assert that REST wins the interoperability battle hands down.

Since one of the driving points behind creating the SOAP specification was to create an interoperable way to communicate between different platforms and different languages, many people are surprised by this assertion. But a funny thing happened on the way to widespread interoperability: the WS-\* specifications (and vendors' implementations of said specifications) made SOAP services less interoperable rather than more interoperable.

The problem in the SOAP and WS-\* arena is the large number of different standards (and versions of each of those standards) to choose from. And when a particular vendor chooses to implement a particular standard, that vendor often provides an implementation that is just slightly different from another vendor's (or all others). This leads to problems whenever you have to cross vendor boundaries (languages and operating system).

Of course, even to use SOAP you need a SOAP toolkit on your platform, which most (but not all) platforms have today. And then you have to deal with myriad WS-\* specifications and figure out which to use (or not to use) and how that affects interoperability. To be honest, it's kind of a mess out there.

In terms of platforms, REST has the advantage because all you need to use REST is an HTTP stack (either on the client or the server).

**But what about metadata? So what if REST is so interoperable—there's no WSDL with REST, and without WSDL, I can't generate a client-side proxy to call a service. REST is hard to use.**

It's true that in the world of REST, there is no direct support for generating a client from server-side-generated metadata, as there is in the world of SOAP with Web Service Description Language (WSDL). A couple of efforts are being made to get such support into REST, one being a parallel specification, known as WADL (Web Application Description Language). The other is a push to use WSDL 2.0 to describe RESTful endpoints. I often say that REST is simple, but simple doesn't always mean easy. SOAP is easy (because of WSDL), but easy doesn't always mean simple.

 SOAP services are much harder to scale than RESTful services, which is, of course, one of the reasons that REST is often chosen as the architecture for services that are exposed via the Internet (like Facebook, MySpace, Twitter, and so on).

Securing Your API – What To Do

<http://blog.smartbear.com/security/api-security-in-rest-vs-soap/>

Over the years, SOAP has added extensions to deal with transactional messaging specific security considerations. SOAP has been around long enough and has been adopted by large enterprises such that it benefits from OASIS and W3C recommendations. Mainly, XML-Encryption, XML-Signature, and SAML tokens help to tighten up the security story over the data being received by and sent from a SOAP service.

REST on the other hand does not implement any specific security patterns, mainly because the pattern focuses on how to deliver and consume data, not how to build in safety into the way you exchange data. Proper amounts of security in code, deployment, and transmission should be determined by those implementing REST architecture patterns, not presumed as something that comes out-of-box.

REST Vs SOAP, The Difference Between Soap And Rest

<http://spf13.com/post/soap-vs-rest>

## SOAP

SOAP brings it’s own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing named operations, each implement some business logic through different interfaces.

Why REST?

Since REST uses standard HTTP it is much simpler in just about ever way. Creating clients, developing APIs, the documentation is much easier to understand.

REST permits many different data formats where as SOAP only permits XML.

REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.

Why SOAP?

## WS-AtomicTransaction

Need ACID Transactions over a service, you’re going to need SOAP. While REST supports transactions, it isn’t as comprehensive and isn’t ACID compliant. Fortunately ACID transactions almost never make sense over the internet. REST is limited by HTTP itself which can’t provide two-phase commit across distributed transactional resources, but SOAP can. Internet apps generally don’t need this level of transactional reliability, enterprise apps sometimes do.

## WS-ReliableMessaging

Rest doesn’t have a standard messaging system and expects clients to deal with communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.

# **REST vs SOAP: When Is REST Better?**

### **The Argument For SOAP**

SOAP is still offered by some very prominent tech companies for their APIs (Salesforce, Paypal, Docusign). One of the main reasons: legacy system support. If you built a connector between your application and Salesforce back in the day, there’s a decent probability that connection was built in SOAP.

There are a few additional situations:

* SOAP is good for applications that require formal contracts between the API and consumer, since it can enforce the use of formal contracts by using WSDL (Web Services Description Language).
* Additionally, SOAP has built in WS-Reliable messaging to increase security in asynchronous execution and processing.
* Finally, SOAP has built-in stateful operations. REST is naturally stateless, but SOAP is designed support conversational state management.

Some would argue that because of these features, as well as support for WS\_AtomicTransaction and WS\_Security, SOAP can benefit developers when there is a high need for transactional reliability.

It is important to note that one of the advantages of SOAP is the use of the “generic” transport.

# **Understanding RESTful vs SOAP web services**

<http://funonrails.com/2014/03/understanding-restful-vs-soap-web-services/>

## REST

#### ighlights:

1. It is simple
2. It is lightweight and faster
3. Platform-independent
4. Language-independent
5. runs over HTTP
6. Response can be returned in “XML/JSON” format

Like Web Services, REST offers no built-in security features, encryption, session management, QoS guarantees, etc. But also as with Web Services, these can be added by building on top of [HTTP:](http://funonrails.com/2014/03/understanding-restful-vs-soap-web-services/)

## SOAP

It defines 3 fundamental properties:

1. What service does: Operations provides
2. How service is accessed: Data format and protocol details
3. Where a service is located: Adress (URL) details

#### Highlights

Here are few **reasons** for which you may want to use SOAP:

Platform and laguage independent

Follows W3C standard  
WSDL (Web services description language) is a W3C standard based on xml and is used to describe web services.

Security  
It supports SSL (just like REST). It also provides a standard implementation of data integrity and data privacy. Being “Enterprise” it’s more secure.

Atomic Transactions  
It supports ACID Transactions (WS- Atomic Transactions) so it can provide two phase commit over distributed transactional resources.

Reliable Messaging   
It has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries

# **WS-Security**

<https://en.wikipedia.org/wiki/WS-Security>

**Web Services Security** (**WS-Security**, **WSS**) is an extension to [SOAP](https://en.wikipedia.org/wiki/SOAP_(protocol)) to apply security to [Web services](https://en.wikipedia.org/wiki/Web_service). It is a member of the [Web service specifications](https://en.wikipedia.org/wiki/List_of_Web_service_specifications) and was published by [OASIS](https://en.wikipedia.org/wiki/OASIS_(organization)).

The protocol specifies how integrity and confidentiality can be enforced on messages and allows the communication of various security token formats, such as [Security Assertion Markup Language](https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language) (SAML), [Kerberos](https://en.wikipedia.org/wiki/Kerberos_(protocol)), and [X.509](https://en.wikipedia.org/wiki/X.509). Its main focus is the use of [XML Signature](https://en.wikipedia.org/wiki/XML_Signature) and [XML Encryption](https://en.wikipedia.org/wiki/XML_Encryption) to provide end-to-end security.

## Features[[edit](https://en.wikipedia.org/w/index.php?title=WS-Security&action=edit&section=1" \o "Edit section: Features)]

WS-Security describes three main mechanisms:

* How to sign SOAP messages to assure integrity. Signed messages also provide [non-repudiation](https://en.wikipedia.org/wiki/Non-repudiation).
* How to encrypt SOAP messages to assure confidentiality.
* How to attach security tokens to ascertain the sender's identity.

The specification allows a variety of signature formats, encryption algorithms and multiple trust domains, and is open to various security token models, such as:

* X.509 certificates,
* Kerberos tickets,
* User ID/Password credentials,
* SAML Assertions, and
* custom-defined tokens.

The token formats and semantics are defined in the associated profile documents.

WS-Security incorporates security features in the header of a SOAP message, working in the [application layer](https://en.wikipedia.org/wiki/Application_layer).

These mechanisms by themselves do not provide a complete security solution for Web services. Instead, this specification is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies. In general, WSS by itself does not provide any guarantee of security. When implementing and using the framework and syntax, it is up to the implementor to ensure that the result is not vulnerable.

Key management, trust bootstrapping, federation and agreement on the technical details (ciphers, formats, algorithms) is outside the scope of WS-Security.

# What is [**non-repudiation**](http://security.stackexchange.com/questions/6730/what-is-the-difference-between-authenticity-and-non-repudiation)

<http://security.stackexchange.com/questions/6730/what-is-the-difference-between-authenticity-and-non-repudiation>

Authentication and non-repudiation are two different sorts of concepts.

* Authentication is a **technical concept**: e.g., it can be solved through cryptography.
* Non-repudiation is a **legal concept**: e.g., it can only be solved through legal and social processes (possibly aided by technology).

Authenticity is about one party (say, Alice) interacting with another (Bob) to convince Bob that some data really comes from Alice.

Non-repudiation is about Alice showing to Bob a proof that some data really comes from Alice, such that not only Bob is convinced, but Bob also gets the assurance that he could show the same proof to Charlie, and Charlie would be convinced, too, even if Charlie does not trust Bob.

Therefore, a protocol which provides non-repudiation necessarily provides authenticity as a byproduct; in a way, authenticity is a sub-concept of non-repudiation. However, there are ways to provide authenticity (only) which are vastly more efficient than known methods to achieve signatures (authenticity can be obtained with a [Message Authentication Code](http://en.wikipedia.org/wiki/Message_authentication_code) whereas non-repudiation requires a[Digital Signature](http://en.wikipedia.org/wiki/Digital_signature) with much more involved mathematics). For this reason, it makes sense to use "authenticity" as a separate concept.

[SSL/TLS](http://en.wikipedia.org/wiki/Transport_Layer_Security) is a tunneling protocol which provides authenticity (the client is sure to talk to the intended server) but not non-repudiation (the client cannot record the session and show it as proof, in case of a legal dispute with the server, because it would be easy to build a totally fake session record)

Nonrepudiation – Dictionary meaning

Nonrepudiation is the assurance that someone cannot deny something. Typically, nonrepudiation refers to the ability to ensure that a party to a contract or a communication cannot deny the authenticity of their signature on a document or the sending of a message that they originated.