

Chapter 13

Security—The Challenges

“The stuff that keeps you up at night is security.”

—Tony Scott, CTO, General Motors¹⁹

When a group of software development managers was asked to identify the obstacles to the deployment of web services, nearly half (47 percent) pointed to security. That was more than twice the percentage of other challenges mentioned, such as bandwidth and access issues (21 percent) and interoperability problems (13 percent).²⁰ While managers justifiably fear hackers and loss of confidential customer information such as they’ve experienced with their e-commerce web sites, web services present even greater security challenges that managers haven’t previously encountered.

The good news is that Internet security technologies for both the World Wide Web and web services have progressed beyond the rocket-science phase. The Ph.D.’s have completed their work, leaving us with well-understood technologies for encryption, authentication, non-repudiation, and trust. The bad news is that we’ve now got to figure out how to stitch these technologies together into an end-to-end security quilt that meets the needs of complex web-services applications. What keeps IT managers up at night is wondering when that quilt will be completed—and whether they should try to move forward with piecemeal solutions in the meantime.

In this chapter, we'll explore the unique challenges of security for web services. We'll begin with the concept of *security contexts*, highlighting the variations in security requirements among different types of web services and the more familiar security technologies of the World Wide Web. Then we'll introduce the building blocks of Internet security, followed by a detailed analysis of the security requirements for web services. In the following chapter, we'll explore the various solutions to these web-services security challenges.

Security Contexts

You'll recall that in Chapter 5 we segregated web services according to their complexity. One of the starkest distinctions between simple and complex web services is the difference in their security requirements. What works well for simple web services (most notably those that are synchronous) doesn't come close to solving the security problems encountered by more demanding asynchronous services.

To get a handle on the differences in requirements between these categories of web services, let's look at the *security context*, or the environment in which a system's security technologies must function. The security context includes two properties or dimensions: space and time. The table in Figure 13-1 summarizes the differences among three security contexts: those of the World Wide Web, simple web services, and complex web services.

	World Wide Web	Simple Web Services	Complex Web Services
Space	In-Transit Only	In-Transit and Multi-Hop	In-Transit, Multi-Hop and In-Storage
Time	Seconds or Minutes	Seconds or Minutes	Perhaps Years

Figure 13-1: Security Contexts

Let's take a few moments to explore this table in more detail, so that we can understand the fundamental differences in the security requirements of the three environments.

Space

Each security context has certain physical boundaries that create a defined space within which information must be secured. As you can see from the table in Figure 13-1, there are three possible locations to consider: in-transit, multi-hop, and in-storage.

In-Transit

While data is being transmitted from one system to another, it's said to be *in transit*. For example, on the World Wide Web the Secure Sockets Layer (SSL) protocol is used to encrypt data as it moves between web browsers and web servers, as illustrated in Figure 13-2.

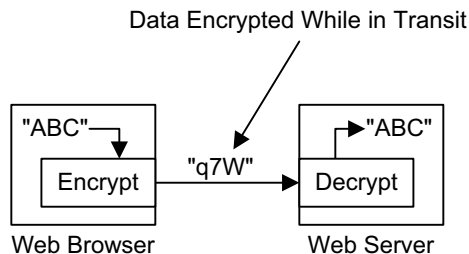


Figure 13-2: In-Transit Encryption via SSL

Using SSL, data is encrypted at one end of the connection just prior to transmission and decrypted immediately upon receipt at the other end. SSL is a *transport-layer* security technology that provides a *point-to-point* encrypted transmission path between two systems. In other words, the security provided by SSL only exists while information is in transit between systems, not while it's stored on the systems themselves. The web's standards and protocols don't address the security requirements of the computer systems on which the browser or server software run—only the links between them, so the web's

security context is limited to in-transit security, as illustrated in Figure 13-3.

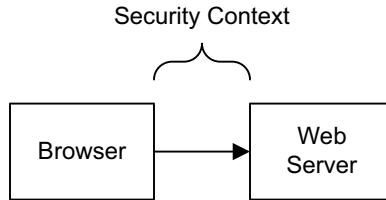


Figure 13-3: In-Transit Security Context for the World Wide Web

The simplest synchronous web services operate in a security context that's essentially the same as that of the World Wide Web, so SSL is often sufficient to meet their security requirements.

Multi-Hop

More elaborate synchronous web services and all asynchronous web services operate in a security context that's substantially broader and more complex than that of the World Wide Web. Specifically, such web services may communicate through *intermediaries*, in which case messages will make multiple *hops* between systems or hosts. This is where we begin to see a divergence between the security contexts of the web and those of web services, and in the ability of SSL to meet the needs of those web services. In a multi-hop topology, as illustrated in Figure 13-4, SSL encrypts and decrypts data each time it's sent over a point-to-point link.

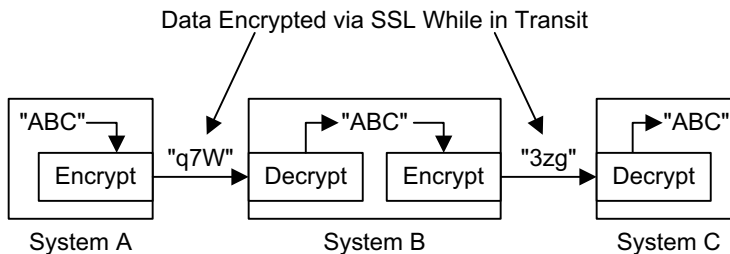


Figure 13-4: SSL for Web Services

There are two problems with using SSL in this application. First, unlike topologies that consist of only two systems, the exchange of XML documents between complex web services may involve a number of stops along the way. Some data should be able to be seen, interpreted, and optionally modified by these intermediate nodes (e.g., System B in Figure 13-4), while other data must remain encrypted and unreadable by the same intermediate nodes. SSL only encrypts data while in transit, so the data is fully decrypted on each intermediate node and is no longer secure. The second problem with using SSL in this case is that if the objective is to deliver data securely from System A to System C, twice as many encryption and decryption operations will be performed than are required.

In-Storage

Complex web services present yet another challenge not shared with either simpler web services or the World Wide Web: the need to secure information while stored as well as while it's in transit. Consider a confidential document, such as a medical record, sent from System A to System C as shown in Figure 13-4. The medical record may need to be stored on System C for later retrieval. For that matter, the record may need to be securely stored on Systems A or B as well. But SSL explicitly decrypts data as it arrives at each new system, so if received data is to be stored securely on any system, it must be re-encrypted via a technology other than SSL.

If a document used in a web-services transaction must be confidential, the web service's *security context* encompasses the various systems on which that document may reside even temporarily, as well as the infrastructure that connects those systems. The greater the number of systems that have access to the document, the broader the spatial scope of the security context, and the more demanding the security requirements.

Time

The security contexts shown in Figure 13-1 are also defined according to time, or how *long* security must be preserved. For the World Wide Web and the simplest web services, data must only be

secured during transmission from one system to another. But for complex web services, data must also be protected during the time it's stored—potentially for very long periods. The complexity of the security requirements increases with a corresponding increase in the time dimension of a security context.

On the web and for simple web services, the time component ranges from a few seconds to a few minutes, so security requirements are comparatively simple and short-lived. A web browser sends a request to a web server, then waits for and receives a response—and that's the end of the relationship between the two entities. If it takes more than a minute or so, a timeout occurs, and the entire process must begin again. Since the time dimension of the web's security context is so short, the security technologies can be relatively simple.

Underlying SSL is the web's HyperText Transfer Protocol (HTTP). Although this is a *connection-oriented protocol*, it only supports very short-lived connections—those that consist of no more than a single request/response exchange. Yet because the security contexts of both the web and simple web services also exist for such short periods of time, both can be based on the primitive security features of HTTP and SSL. The short-lived transport-layer connections between browser and web server or between web-services requestors and providers are sufficient for the brief duration of these applications.

When a somewhat longer-term relationship is required on the web, it can be managed through the use of *cookies* or other means, but no comparable standard exists for web services. And truly long-running asynchronous web services require that security be *persistent* and maintained over an extended period of time. This is a challenge rarely encountered in simpler World Wide Web applications, which is another reason why the security requirements for complex web services exceed the solutions offered by the web's existing standards and protocols.

Security for Asynchronous Web Services

The security context matrix has helped us see that the security requirements of simple web services are similar to those of the World Wide Web, and that many simple web services' security requirements

can be met by using standard web protocols. HTTP and SSL provide a shortcut for simple web services that have the following attributes:

- They involve only two entities or endpoints.
- The entities are only connected for relatively short periods of time (seconds or minutes).
- All that needs to transpire between the two entities can be accomplished within the context of those short-lived connections.

For the remainder of this chapter, we'll take on the more difficult security challenges of external, asynchronous, and aggregated web services, beginning with an analysis of the building blocks of security.

The Building Blocks

Security is a broad topic, but it can be broken down into five very specific elements or building blocks, as illustrated in Figure 13-5. These building blocks are applicable to virtually all data-processing environments, including the World Wide Web and web services.

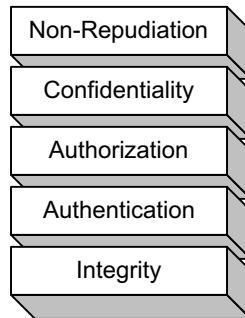


Figure 13-5: The Building Blocks of Security

- **Integrity** ensures that documents, messages, and their components have not been altered.
- **Authentication** guarantees that an entity (a person or system) is who or what it claims to be.
- **Authorization** determines the privileges available to an authenticated entity.

- **Confidentiality** ensures that unauthorized parties can't read documents, messages, or their components.
- **Non-repudiation** prohibits an entity from denying that it sent or received a message.

Integrity

Perhaps no other concept is as fundamental to security as our ability to know for certain that the documents and messages we receive haven't been altered, either maliciously or due to technical errors such as packets damaged in transit. Credentials presented by a consumer or business partner can't be trusted unless you're confident they haven't been forged. Likewise, a digital receipt is of little value unless it can be shown to be tamper-proof. Without integrity there can be no authentication, authorization, confidentiality, or non-repudiation.

SSL is a sufficient solution for the integrity of simple synchronous web services where the security context consists of only a single pair of endpoints, and the relationship between them lasts for no more than the time it takes to exchange a single pair of request/response messages. But SSL is inadequate for complex web services, whose unique requirements include both *end-to-end integrity* and *component-level integrity*.

End-to-End Integrity

SSL can ensure the integrity of information between a single pair of entities, but not if the information must pass through one or more intermediaries. Once data has been decrypted on an intermediate system, SSL can no longer guarantee the integrity of the original data. This is the fatal flaw of using transport-layer encryption to try to guarantee the integrity of data in any but the simplest of web-services architectures. Even some synchronous web services make use of intermediaries, so SSL may not be adequate even for them.

Component Integrity

Transport-layer encryption provides all-or-nothing integrity. In this case, a stream of data containing packets, messages, and documents

is decrypted in its entirety, so there's no way to allow an intermediary to modify one portion of a message or document while prohibiting that intermediary from modifying other portions. In other words, if any of the data can be altered, there's no way to keep other data from being altered as well.

Intermediaries can be used to perform *transformations* that intentionally modify portions of messages or documents. For instance, a transformation service might convert an invoice amount from US dollars to Euros. However, this intermediary should not be allowed to modify other components or elements of the message. Web services must be able to guarantee the integrity of information at multiple levels, such as the following:

- **Documents.** In some applications, integrity should be maintained at the document level. For example, a contract or other traditional document should be guaranteed intact.
- **XML elements.** Some applications require that the integrity of individual elements within an XML document be maintained separately. This allows some elements of the document to be modified by intermediate processes, yet guarantees protection to elements that must not be changed. For instance, as documents are routed through various stages of electronic approval, the documents may accumulate digital signatures, but the original content should not be altered.
- **SOAP messages.** XML payloads may be carried within SOAP messages, and in some instances integrity must be maintained at the SOAP level rather than for each individual document.
- **Digital credentials.** The integrity of usernames, passwords, and digital certificates must also be guaranteed, sometimes independently of the documents and messages to which they apply.

Authentication

After a foundation of integrity is established, authentication is the next building block for web-services security. Authentication allows web-service requestors and providers to verify the identity of the entities with which they interact.

Username and passwords are by far the most common form of authentication on the World Wide Web, and this web-based approach to authentication may meet the needs of particularly simple web services. But many web services have requirements that exceed what's available from the web's security mechanisms. Five authentication requirements for web services go beyond what we typically encounter on the web:

- Loosely coupled authentication
- Bi-directional authentication
- Credential consolidation
- Multi-party authentication
- Durable authentication

Loosely Coupled Authentication Models

Web-services endpoints that are owned by different entities will probably have their own models, systems, and standards for authentication. One system may be based on Kerberos (an authentication system developed by MIT and used within Microsoft's .NET), while another may use *public-key infrastructure* (PKI). In order for these web services to work together, there must be a trusted mechanism by which the disparate models can exchange identities. And because authentication models can vary so greatly, such a system must be loosely coupled just as web services themselves are loosely coupled. Well-designed web services must be flexible in their expectations of other systems' authentication models.

Bi-Directional Authentication

Authentication is accomplished when one entity presents its *credentials* to the other, as illustrated for the World Wide Web in Figure 13-6.



Figure 13-6: Consumer Use of Credentials

E-commerce web sites authenticate consumers through a combination of usernames, email addresses, passwords, and other schemes, and many simple web services can use these same techniques. But authentication in the opposite direction is rare on the web. Although server-side digital certificates are always used in conjunction with SSL, few web-site visitors bother to check them. In fact, few consumers know how to verify the identity of the sites they visit, and even fewer do so as a matter of course.

There have been many cases where high-visibility sites have been hijacked, and web-site content delivered from unauthorized servers. Sometimes going to the wrong web site is a simple user error—for example, www.whitehouse.gov is an official site of the President of the United States, whereas www.whitehouse.com is a porn site. In the case of consumer e-commerce, the risks due to weak authentication are mostly embarrassing rather than costly.

For web services the need for bi-directional authentication (as illustrated in Figure 13-7) is more critical.

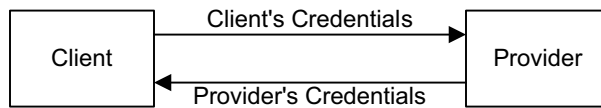


Figure 13-7: Bi-Directional Authentication

There are three reasons why this bi-directional authentication is required for web services. First, the consumer-protection laws that apply to individuals using their credit cards online don't similarly protect businesses that utilize web services. A US consumer's liability for fraudulent purchases made using his or her credit card is limited by statute to US\$50. No such limits exist in the US for businesses-to-business transactions.

Second, the values of business-to-business web-services transactions are typically much greater than those in business-to-consumer commerce. Such values can be either monetary or derived from the confidential nature of information being exchanged, since the unauthorized or inadvertent publication of a trade secret can be very expensive.

Third, web services are based on unattended automated systems, so there's an increased risk that damage may go undetected. Whether due to an error or a malicious attack, automated web services that run amok can create substantial liabilities. When using web services, bi-directional authentication can be critical.

Credential Consolidation

In the typical multi-tiered architecture used by e-commerce web sites and web services alike, consumers and requestors don't interact directly with back-end systems such as databases or legacy applications. Instead there's typically at least one intermediate system—often an application server or *portal*—that accepts requests from consumers and communicates with the database or other back-end system on their behalf. In this sense, the application server is acting as an *agent* of the requestor or consumer, as illustrated in Figure 13-8.

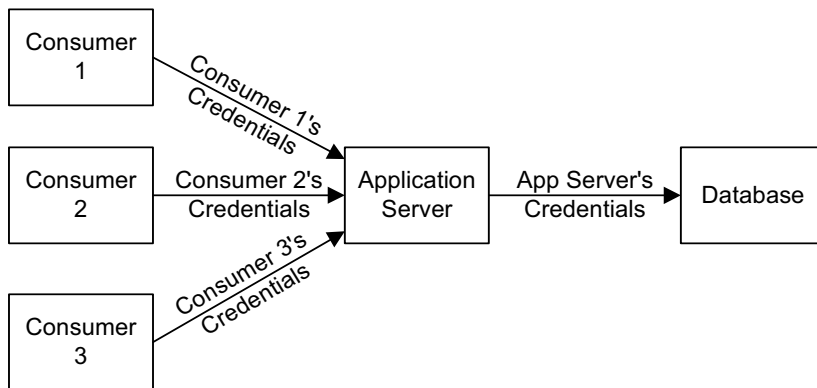


Figure 13-8: Credential Consolidation

For its own protection, a back-end database system must authenticate its users—but in the above example, the actual *user* of the database is the application server rather than a consumer. The typical solution to this problem is to develop business logic within the application server to perform both the authentication and authorization functions (in other words, to authenticate the consumers and also to determine their privileges). The application server must have broad *superuser* privileges on the database system, and then parcel out these

privileges on a consumer-by-consumer basis. In this instance, the authorization logic is located within the application server—which is not particularly appropriate if authentication must be performed in other locations as well. This also creates an additional vulnerability: If a hacker can gain access to the application server, he or she will then have superuser privileges to read and modify the database.

Furthermore, this architecture destroys the ability of the database to discriminate or even identify individual consumers. The application server can pass along the consumer-identifying data to the database, but this requires a custom application on the database system to enforce consumer-specific authorization and authentication policies. Such one-off applications can be difficult and expensive to maintain.

Multi-Party Authentication

When web services are aggregated, it results in a problem similar to the one faced by aggregators on the World Wide Web, such as travel-reservation services. When an individual consumer uses a web browser to visit a travel aggregator's web site and make airline reservations, there are actually three authentication operations required, as shown in 13-9.

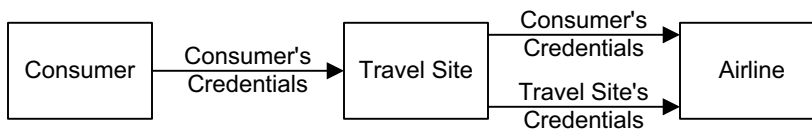


Figure 13-9: The ID-Passthrough Problem

Not only must the consumer authenticate him- or herself to the travel site (to guarantee payment, for example), but the travel site must also present credentials for itself *and* pass-through the consumer's credentials to the airline. The travel site must provide the consumer's name, address, and frequent-flyer numbers, and it must authenticate itself in order to receive its commission. When aggregated web services are built upon many such relationships—linearly or hierarchically—the challenges of multi-party authentication become increasingly complex.

Where to Provide Authentication Services

After deciding what to authenticate, the second question an implementer of web services must answer is: At what point in the architecture should authentication be provided? Previously, we looked at a model where the application server was responsible for authentication. On the World Wide Web, this function is performed either in the transport layer (using the authentication facilities built into HTTP) or within the custom-written web applications. Neither of these solutions is ideal, even for e-commerce web sites. But although we've become accustomed to them both as consumers and as developers, a better and more standardized solution for web-services authentication is required. (We'll look more closely at the question of *where* to provide security solutions in the next chapter.)

Durable Authentication

Another requirement for web-services authentication is *durability* or *persistence*. Using the mechanisms built into HTTP, authentication is valid only for a single request/response exchange. Although usernames and passwords need not be re-entered, they're re-transmitted by the browser or web-services requestor each time a request is made. Due to the *stateless* nature of the HTTP family of transports, there is no association of one request/response exchange to the next, and each subsequent request must therefore include the authentication credentials.

When it comes to asynchronous web services, there's a need for authentication to persist far longer than the time during which two endpoints are communicating. In our example of the online bookstore in Chapter 8, the session during which the customer's credentials were presented was terminated once the order had been placed. However, those credentials must be accessible for as long as the merchant retains the order information, perhaps for many years.

This presents two challenges. First, the authentication credentials must be retained along with the documents and messages that move through the web-services pipeline. (In some cases, the credentials will be contained within the documents.) Second, there must be a mechanism for verifying credentials long after they're initially presented, and

it must work for an application that may not be able to connect to the system that originated the transaction.

A web site's server can query a user for additional authentication information at any time, because the browser and server remain connected for the life of the session. But asynchronous web services often need to verify credentials or perform other authentication tasks long after the consumer or web-services system involved with the transaction have been disconnected. This is also a requirement for non-repudiation, since it may be necessary to re-establish the authenticity of the parties months or years after a transaction has been completed.

Authentication is made even more difficult when there's a break in the chain of trust—for instance, when a digital certificate expires or an intermediate *certificate authority* goes out of business. These are very real challenges in situations where credentials must be stored and retained for extended periods.

Authorization

Once an entity's identity has been authenticated, the next step is to determine what that entity is authorized to do. Some of the authorization challenges faced by web services include the need for loosely coupled authorization models, authorization durability, identity consolidation, and service-level authorization.

Loosely Coupled Authorization Models

Two endpoints controlled by different entities will probably have their own models and systems for authorization as well as authentication. One system may simply associate usernames with directory and file privileges, while another might depend on a more elaborate rules-based system for authorization. In order for these web services to work together, there must be a mechanism by which the multiple models can exchange identities and mediate their authorization-concept differences.

Durable Authorization

Because complex asynchronous web services may involve long-lived transactions—particularly those supporting external business processes—it's quite possible that authorizations and permissions will change over a transaction's lifetime. This presents a number of challenges that can be difficult to resolve. For example, when a customer's purchasing limits are lowered, is that customer allowed to increase the quantities on already-approved orders? It's one thing to code the business logic that implements such policies within a stand-alone application, but it's far more difficult to do so in the distributed asynchronous environment of web services.

Identity Consolidation

From our discussion of credential consolidation, you'll recall that in many application environments, the identity of the individual user is lost when back-end system access is performed through an agent. The problem is compounded for authorization, as it's no longer possible to make decisions about user privileges once a user's identity has been lost or consolidated.

Service-Level Authorization

Ultimately, web services will require an authorization model of their own, beyond what currently exists for the World Wide Web. Such a system must determine who has access to what services, and within the context of an individual service, what those individuals are allowed to do. Evolving web-services protocols address these unique requirements, determining which specific individuals or other services are allowed to execute certain methods and which are authorized to modify specific XML elements.

Confidentiality

By combining the building blocks of integrity, authentication, and authorization, we can create confidentiality: the ability to ensure that documents, messages, or their components can't be read by any

other than authorized entities. Integrity gives us the knowledge that information hasn't been intentionally or otherwise altered; authentication allows us to identify entities; and authorization lets us determine whether those entities should be allowed access to the confidential information.

Confidentiality is often confused with integrity, and it's true that the two are very closely related. But integrity only allows us to determine whether information has been modified, and only coincidentally keeps that information out of the hands of unauthorized entities. Encryption guarantees integrity while prohibiting information from being understood by unauthorized entities, but it's quite possible to guarantee the integrity of data without encrypting it. One such example would be when a digitally signed document is sent as clear or unencrypted text, which can be read by anyone who intercepts it but can't be altered without detection.

On the World Wide Web and for simple synchronous web services, transport-layer encryption (SSL) is the most common way of maintaining the confidentiality of information in narrow and short-lived security contexts. But as we've seen before, the limitations of SSL quickly become apparent when applied to the broader, long-lived security context of asynchronous web services.

Using SSL, once data is received at its ultimate destination or at an intermediate location, it's restored to its unencrypted format. Data stored on disk or even retained in RAM can no longer be considered confidential, except to the extent that it's protected using additional methods.

For example, if you download account information from a banking web site using SSL, it will be protected while in transit. But if you save that information on your disk drive, it becomes as vulnerable as any other data stored there. Protecting data stored on users' systems or on a web server falls within the domain of security techniques that are beyond the scope of SSL, and hence require more sophisticated solutions.

Web services have confidentiality requirements that extend beyond what SSL provides and fall into four categories: end-to-end encryption, transport independence, encrypted storage and element-level encryption.

End-to-End Encryption

This challenge is similar to the problem of integrity. In the same way that SSL can't guarantee that data hasn't been modified on intermediate systems, it also can't protect that data from unauthorized access. Data is unencrypted—and therefore accessible—on each intermediate system between the endpoints.

As we saw in Figure 13-4 earlier in this chapter, any data flowing from A to C is temporarily decrypted while on B. A and C can't depend on SSL for confidentiality, since SSL doesn't support end-to-end encryption. The solution is true end-to-end encryption, as illustrated in Figure 13-10.

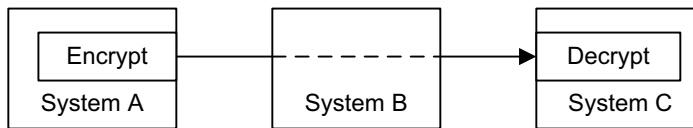


Figure 13-10: End-to-End Encryption

End-to-end encryption can't be implemented in the transport layer. Instead, it must be implemented either within the applications that run on Systems A and C—or better still, in standard software libraries or hardware or software firewalls. (We'll look at these options in the next chapter.)

Transport Independence

The World Wide Web uses HTTP(S) over TCP/IP as its transport protocols, but web services aren't tied to a single transport protocol. Since a web service might use HTTP for one exchange and SMTP or FTP for another, it's inappropriate to implement encryption uniquely within each of these protocols. The solution should be moved up the protocol stack, so that encryption is applied to web-services documents and messages rather than to the transport protocols that carry them.

Encrypted Storage

Because SSL only encrypts documents and messages while in transit, those messages must be re-encrypted using another technology if they're to remain confidential while stored. Looking again at Figure 13-4, let's suppose B acts as a third-party auditing web service, maintaining a log of messages between A and C. While it's important for B to store copies of the messages, it may be inappropriate for B to be able to read them. An end-to-end encryption scheme will prohibit B from decrypting the messages, but allow it to store them in their encrypted form.

One common mistake is to assume that storing information in an encrypted form is the same as limiting access to that information. Encryption is a helpful tool, but it isn't the complete solution. If hackers can reach your information, encryption can keep them from reading it, but there are also two important reasons why you should take reasonable steps to keep the hackers from getting to the information in the first place.

First, even if they can't exploit the encrypted information, they might be able to do harm in other ways, just by virtue of being inside your systems. For example, they might find a way to delete an important document, even though they can't read it.

Second, allowing hackers to gain the knowledge that a document or file exists may be enough to cause you serious harm. For instance, a personnel document may be encrypted, but if the file name happens to be the employee's US Social Security Number, hackers can identify employees and collect valid SSNs. (But no one would be so foolish as to design a system that used SSNs as filenames, right? Wrong! It happens.)

Element-Level Encryption

Because simple web services involve only two endpoints (the requestor and the provider), there's no need for the individual elements within the messages that pass between them to be encrypted individually. These web services can use SSL or a comparable transport-layer encryption scheme, even though such a scheme has no awareness of the structure of the data it encrypts. Transport-layer encryption indiscriminately makes *all* data inaccessible to third parties. It's a brute-

force approach, with no way to let selected participants gain access to some portions of a message but not to others. 13-11 illustrates a web-service message *envelope*, which in turn contains a *header*, a *body*, and (within the header or body) individual *elements*.

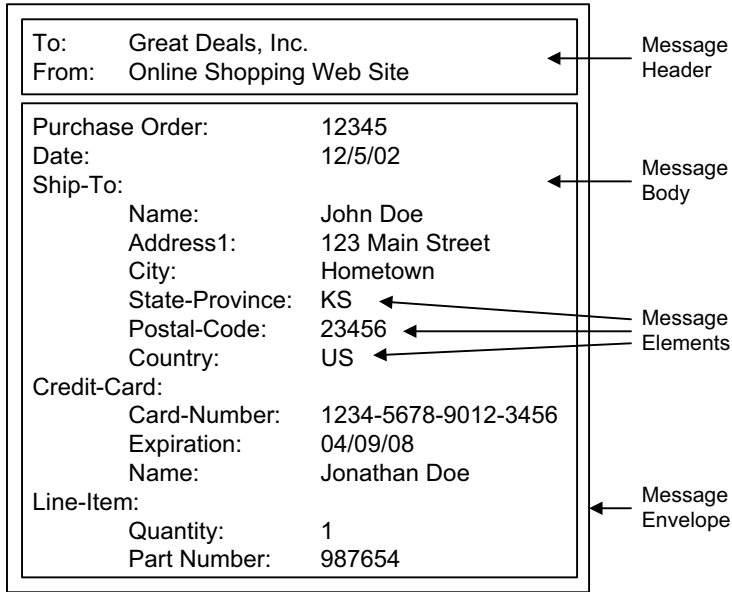


Figure 13-11: Web-Service Message

Using SSL or another transport-layer encryption scheme, a message such as this would be encrypted in its entirety, but only while in transit between two nodes. This is a problem for multi-hop web services, which often need to protect the elements or fields of web-service messages individually. Consider the requirements of an aggregated web service, as illustrated in Figure 13-12.

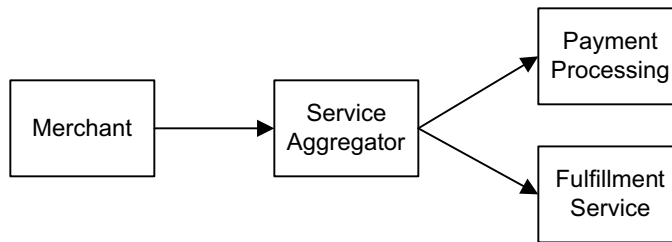


Figure 13-12: Web-Services Message Flow

The merchant's system creates an order such as the one shown in Figure 13-11, and forwards it to the service aggregator. The aggregator's system must be able to read and understand most of the message, but there's no reason it should have access to the consumer's credit-card information. The aggregator should merely forward that data as-is to a payment-processing service. Therefore, the merchant uses *element-level encryption* to keep the consumer's credit-card data confidential end-to-end, or all the way through to the payment-processing service. Element-level encryption protects the individual elements or fields within a web-services message, as illustrated in Figure 13-13.

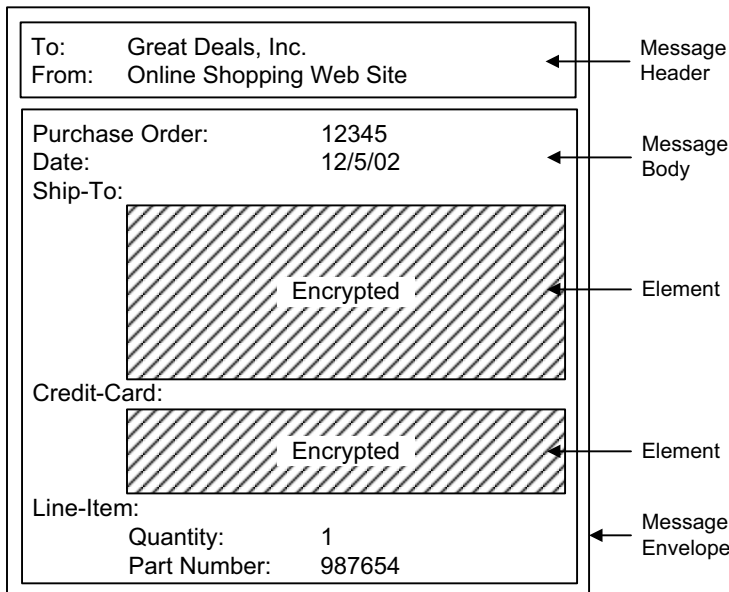


Figure 13-13: Element-Level Encryption

The merchant encrypts the ship-to data in a manner that it can be read by the aggregator (who perhaps handles customer service, and therefore needs to know the shipping address), by the fulfillment service, and by the payment processor. However, it encrypts the credit-card data in such a way that it can be decrypted only by the payment-processing service. The message passes through the aggregator's system, and can even be safely stored there in case it's required at some time in the future. Likewise, the aggregator forwards the message to both the fulfillment and payment services, but only the payment processor has access to the credit-card data.

Non-Repudiation

SSL or other transport-layer encryption schemes can be used to meet many of the security needs of simple web services, much as they do on the World Wide Web. But SSL has no built-in capability for non-repudiation: proof of events, such that (for example) buyers can't deny placing orders, and sellers can't deny receiving them.

The existing non-repudiation methods on the World Wide Web have been implemented by custom applications or in some cases using proprietary packages. The most familiar non-repudiation technique would be a printable receipt delivered as a web page, although this isn't particularly strong in a legal sense because it can easily be forged. Receipts offer a degree of protection for consumers, but most merchants simply rely on credit-card fraud detection to minimize risk and absorb the cost of whatever fraud can't otherwise be prevented.

Some web services can get by without a non-repudiation strategy. For instance, web services that don't include commercial transactions typically don't require non-repudiation. If you query Amazon.com for the price of a book, there's no need for either you or Amazon.com to be able to prove that the event occurred. But many commercial transactions—even those conducted over simple synchronous web services—require non-repudiation for the legal protection of the business entities involved.

The techniques for establishing non-repudiation using digital certificates are well understood, although they haven't yet been uni-

versally and consistently adopted. We'll explore these solutions in the following chapter.

Defensive Security

We can't conclude our discussion of web-services security without considering the vulnerability of web services to attacks, and the measures required to defend against such attacks. Although web-services systems are susceptible to many of the same attacks as those that occur on e-commerce web sites, we'll focus our attention on the vulnerabilities unique to web services.

Denial of Service (DoS)

Network firewalls and other tools can be used to detect and block DoS attacks—attempts to disable a service by flooding it with traffic—at the network and transport layers. But web services are also vulnerable to application-layer DoS attacks. It's not enough to merely watch for packets sent to particular ports or carrying payloads based on one protocol or another. Web-services defensive systems must employ application-layer logic, such as looking for sudden increases in the number of transactions per unit time or tracking high-level business metrics (e.g., the total dollars per hour that are processed by a service).

Replay Attacks

Another application-layer attack is referred to as a *replay attack*, and occurs when a hacker captures and then repeatedly re-submits a transaction request. The damage can be similar to that caused by a DoS attack, when the system rejects the forged transactions but gets bogged down doing so. Worse still, the forged transaction may actually be accepted and acted upon. The best protections against this are strong authentication, document or message integrity, and the unique (and encrypted) identification of all transactions.

Downgrade Attacks

Security downgrades aren't attacks in the traditional sense, but they pose a risk similar to that of any explicit attack. In its simplest form, a downgrade attack occurs whenever you interact with a business partner whose security policies or practices are less robust than your own. Once you send data to a partner, the information is only protected to the extent of the partner's own security policies and practices—even if that information is encrypted. If your partner's environment isn't as secure as yours, the level of security will effectively be *downgraded* simply by virtue of your data being shared with a less robust partner. Furthermore, what guarantees do you have that your partner's security policies or practices won't change over time? A partner that meets your security requirements today may not do so tomorrow.

Now consider a scenario where your data passes through a middleman or aggregator, such as illustrated in Figure 13-12. Even if you have agreed to strong levels of authentication, encryption, and so on, the aggregator may have far less robust relationships with other entities with whom your data will be shared (such as the fulfillment service or the payment processor). Again, you may be subject to a security downgrade attack, and this may occur without your knowledge or approval.

Given all the challenges of web-services security, perhaps it's no wonder it keeps IT managers up at night. As complex as this landscape may appear, there are solutions on the horizon, and now that we've detailed the security challenges for web services, we'll turn our attention to exploring and comparing the variety of security solutions in the chapter that follows.