Global Information Assurance Certification Paper

Integrating Real-Time Services on the Web

Pete Kobak

GSEC Practical Assignment (v1.4b option 2)

March 26, 2003

Abstract

A large financial institution had accelerating needs in 2000 to introduce new business services on its web site. Some services had to be provided by external organizations. The institution previously used leased data lines to provide an external service, but that communication method became too costly to use for many new services. Along with other team members, the author created or defined processes and methods to use the Internet for those communications. This paper describes the development of technical processes and analysis models that enable the institution to quickly and safely integrate new business services into the institution's web site.

Outline

Integrating Real-Time Services on the Web

Preface

For brevity and anonymity I refer to the **s**ubject **f**inancial **i**nstitution as "SFI" throughout this document. This and other definitions used by this paper, particularly TLAs, can be found in the Glossary.

SFI is a large institution, and there are very few significant technical decisions that are made by individual contributors. That is certainly the case with this subject; that's why I use the word "we" for almost all the decisions described here. It was a team effort, where the team sometimes extended to the vendors with whom SFI was trying to integrate. However, the specification for the GSEC *practical assignment* requires the writer to identify his or her own contributions:

I was a significant contributor to all the designs described here. I had a key role in most of the lower-level design decisions, such as key strength and algorithm choices. I worked to make sure that our PKI certificate processes rely on one-to-one off-line validation and real-time SSL authentication. I personally wrote some of the initial code for client-side SSL connections, random number seeding, and anonymous user identifier generation. I wrote all of the command-line tools for certificate handling.

This document is not intended to be an historical record. The actual decision-making process was messier than I imply here, but the conclusions and reasons for the decisions are accurate.

Before

Similar to other financial institutions, SFI's presence on the web transitioned around 2000. Before that, the web was used for education and for self-service of the core financial services of SFI. In early 2000, the web became more strategic in enabling SFI to offer a full range of financial services at relatively low cost. The business need to offer wider services outside of SFI's core competency required SFI to engage external *application service providers* (ASPs).

An ASP provides a service to individuals via a web browser interface. SFI contracts with an ASP to provide those services to SFI's customers. In many cases, the service is tightly integrated into SFI's web site such that the experience of using the ASP is as seamless as possible. In some cases, the service integration is looser so that the company or brand of the ASP is exposed intentionally.

Single Sign-On

The biggest challenge to integrating the user's experience is *single sign-on* (SSO):

1. A user authenticates on SFI's site using username & password credentials.
2. The user navigates to a link to the ASP's service and hits the link.
3. The user is presented with the first page of the service at the ASP web site, without entering credentials.
4. The user continues to use the service at the ASP, and eventually ends the session at the ASP.
5. The user returns to using the SFI site.

Some writers use the term *single sign-on* to refer to global identity confederation efforts like Microsoft's *Passport* or the Liberty Alliance. I am applying the term narrowly; only to uses between two web sites, in particular between SFI's site and its ASPs' sites.

Although loosely coupled ASPs can use separate credentials, tight integration requires SSO for a seamless user navigation experience. The implications of SSO include a shared identification reference, secret identification exchange, and that the ASP must trust SFI's authentication.

Private Channels and Public Network

SFI had a single SSO ASP. The method used for this ASP passes all user page requests for the ASP through SFI first; we call this the *proxy* SSO method. SFI adds a user identity number to the HTTP header before passing the request on to the ASP. Because every request has to pass through both SFI's and ASP's servers, and make an additional network hop, the proxy method has disappointing performance.

To improve performance, and to assure confidentiality at a time (1998) when SSL was more difficult to deploy than today, SFI used a private data circuit to the ASP. While this simplifies the technical problem from SFI's perspective, the method has two substantial problems. The first problem was the ongoing telecommunication charges for maintaining the circuit. The second problem was the inability to accurately predict the time needed to install a new circuit. At the height of the "dot-com" boom (1999-2000), it was difficult to order new telecommunications services on a reasonable and predictable schedule.

When several external services were required in a short period in 2000, it became obvious that a new method was needed to improve performance, cost, and time to market.

Models and Frameworks

Although we concentrated on getting the basics right for the first two ASPs, we also needed a longer view to use frameworks in which to create new information flows, and models for evaluating new ASP processes. There was also a need to standardize technical services, especially cryptographic services, and enforce technical policies (such as encryption and key strength) through technical means.

Little Crypto Experience

SFI's experience in cryptography was limited to using crypto-based products like web servers and Kerberized Unix logins. No one had attempted to use crypto directly. I had done some paper research on the use of client-side web browser certificates as an alternative to username/password authentication of web site users, so I knew a bit about PKI, asymmetric encryption, and digital signatures. However, no one I knew of at SFI had actually written code for SFI using crypto directly.

During

This section describes my efforts, in conjunction with many others, to meet the needs described in the *before* section.

Token Single Sign-On

The basic requirement was to define a technique for SSO that requires only Internet connections. A big plus would be to allow communications directly between the end-user's browser and the ASP, significantly increasing performance over any scheme that involves SFI being in the middle.

Basic Information Flow Requirements

To allow a user to be authenticated at SFI, and then engage directly with an ASP, the user's identity must be passed to the ASP. To establish a cookie-based session between the user's browser and the ASP, the user's browser must make a request directly to the ASP. Taken together, these two requirements add up to having to pass the identity from SFI, through the browser, then to the ASP:

1. User logs on to the SFI site, using credentials known only to SFI.
2. SFI creates a server-based session for the user.
3. SFI sends a cookie to the user's browser to link to the session.
4. This cookie serves to identify the user to SFI on subsequent requests to SFI.
5. User navigates to a link to access the service eventually served from the ASP.
6. The user's identity is passed through a form field to the user's browser.
7. The user's identity is passed from the form field to the ASP.
8. ASP authenticates the user based on the identity from SFI via the browser.
9. ASP creates a server-based session for the user.
10. ASP sends a cookie to the user's browser to link to the session.
11. This cookie serves to identify the user to ASP on subsequent requests to ASP.
12. The user continues to access ASP web pages.
13. The user can also continue to access SFI web pages.

Having created the necessary series of basic steps, we proceeded to design the identity-passing mechanism.

Token Design

Because we knew we needed to perform crypto operations on the identity (although we hadn't determined the operations yet), we knew the information passed from SFI to ASP had to be encapsulated in some way. We began to use the term *token* to describe this encapsulated identity data. We also used the term *token* to refer to the general SSO method, in contrast to the *proxy* SSO method.

The user's identity is the primary payload in the token. Although a real-world identity (name, address, SSN, etc.) could be used if necessary, we were always able to use a number without a real world meaning. I'll describe that number a bit more below, but at the time we assumed a single number would be sufficient.

Beyond the user's identity, additional information may be required to operate the service. For instance, the identity of the source of the token may be necessary to allow the ASP to distinguish between multiple service integrators besides SFI. Information about the user, in addition to the user's identity, may also be required. Altogether, the main payload of the token has to be flexible and self-describing.

Although we hadn't settled on encryption or signing techniques, we soon realized that no matter how the token was protected by crypto, a replay attack would be possible when the token is on the user's browser. In other words, the token could be reused by an unauthorized user if, despite proper HTTP page cache settings, the browser or other components of the user's computer allows the token to be recovered after use. To prevent this, a timestamp was added to the token to indicate when the token was created.

To use the timeout, two agreements with the ASP are necessary. First, a time synchronization method must be agreed to between SFI and ASP. In practice, this is accomplished by establishing the tree of time authorities to a common source, and estimating the theoretical time drift between SFI and ASP. Second, the time-to-live or timeout period is agreed to. This should be the maximum expected time from token creation to token receipt including network, server, and

browser delays; plus the maximum time drift between servers as determined by the analysis of time synchronization.

Sometime after initial tests, SFI and an ASP realized that it was important to establish a correlation between log entries on both sides for debugging and possible non-repudiation. Because of relatively low volume of use compared to the timestamp precision, that requirement was met in practice by the combination of user identity and timestamp values. However, if we couldn't reasonably assure uniqueness of those fields, we would have had to add an explicit field to the token.

Many months after the original token specification, and after we had studied crypto a bit more, we added a purely technical field. Because the other fields of the token were relatively predictable, we added a random number to the token to increase its entropy.

The physical layout of the token sometimes varied between ASPs. Initial ASPs were starting from scratch like SFI was, so that the token layout was a de-facto standard. Later ASPs sometimes already had a layout defined to which we sometimes had to add fields to meet SFI's standards. Our preferred field demarcation was XML, but some ASPs asked us to use proprietary layouts.

Logical field summary:

Classification	**Logical Field**
Core Payload	
	SFI identifier
	User identity number
	Additional user information
Token Protection	
	Timestamp; usually milliseconds from Unix epoch
	Unique token number if necessary; usually not needed
	Random number

Anonymous User Identifier

The core of the token is the identity number for the user. The number has to unambiguously identify the person authenticated at SFI. But what number should be used?

When we first looked into solving this problem, the obvious choice was Social Security Number (SSN). It is the closest thing the U.S. has to a universal identity number. However, it is also subject to abuse and legitimate privacy concerns. We threw out the use of SSN even though it would have made things a lot easier for SFI and its ASPs. This decision was prescient because in the years since, SFI has adopted more stringent SSN usage policies in line with heightened privacy concerns and the introduction of SSN protection laws.

In the first two ASPs, we used a number generated just for the shared identity of the user. The ASPs did not need to know the real-world identity of the user. They just needed to consistently refer to the same person, and of course to have the additional information they needed to provide the service. At the time, we thought this was a happy but exceptional case, and we thought future ASPs would have to know the user's identity. After integrating many more ASPs, it turned out that keeping users anonymous to the ASPs was the rule rather than the exception. This is an important privacy enhancement, and SFI has worked hard to minimize the user identity information required by their ASPs.

Three of the methods we used to generate anonymous user identifiers, in decreasing order of desirability:

1. Generate a salted hash of an internal SFI user identifier. This has the advantage of being strong and not requiring storage. It is regenerated each time the number is

required. Because it is not available for lookup, this method can only be used if the ASP doesn't have to ask for additional information about a user.

2. Generate a unique pseudo-random number to identify the user for each ASP. This is also a strong method that can be used where an ASP needs to ask SFI about a user. The disadvantage over the first method is the need to store the identifier for M users times N ASPs.
3. Use an existing internal SFI user identifier. This is a weaker method than the first two, but is the cheapest and easiest of the three. The internal identifier has no meaning in the real world, nor can it be used through un-trusted channels into SFI. This method is used when there is no other user identity information flowing to the ASP.

Token SSO Flow

Having described both the intended SSO flow and contents of the token, it is useful to show the flow graphically. The steps shown in Figure 1 are simplified compared to the list in the Basic Information Flow Requirements. Sequence numbers in this diagram correspond to a narrative in the Appendix. The token is represented as the "Access" rectangle.

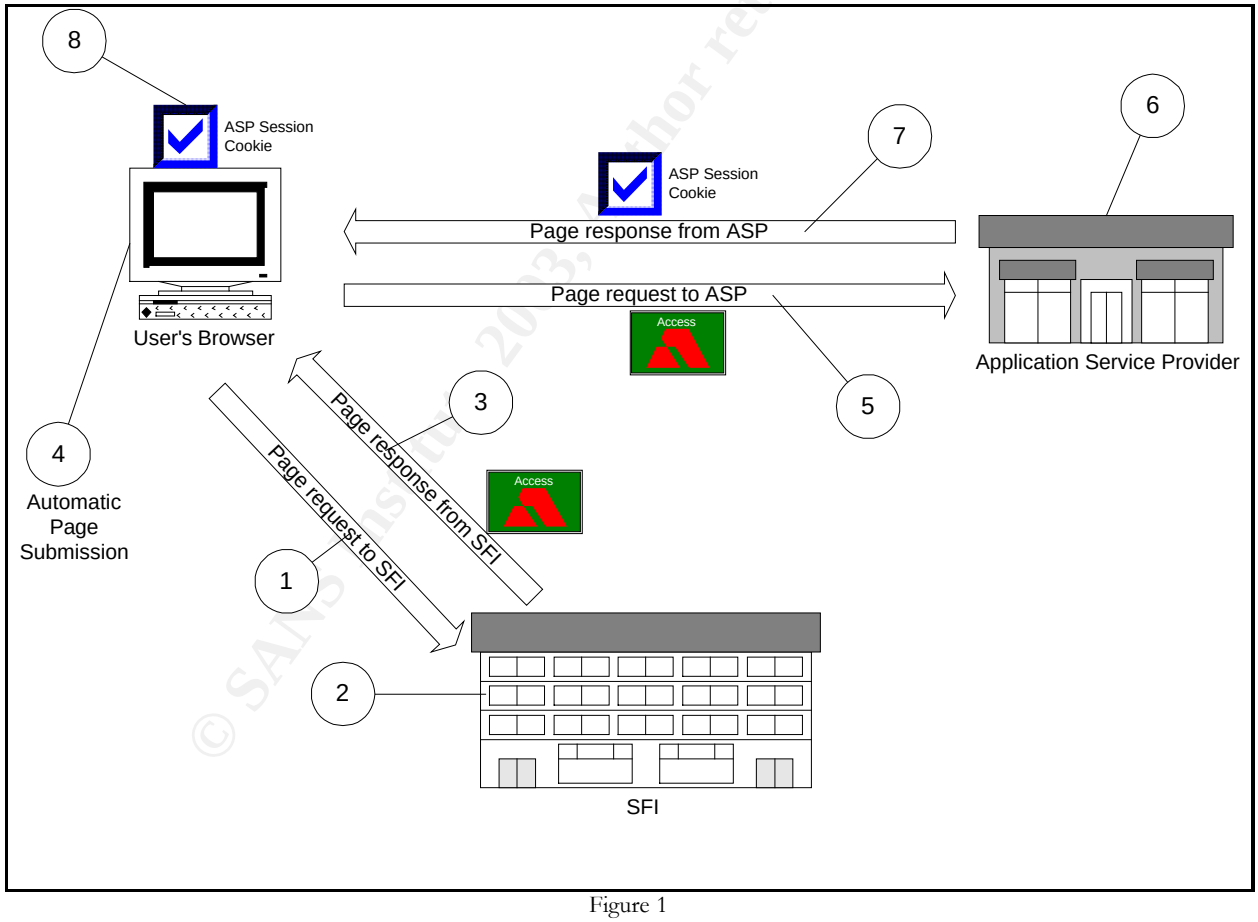

Figure 1

The automatic page submission shown in step 4 (in Figure 1) is a Javascript-initiated HTTP POST to the ASP. When we were first designing the flow, we thought we would have problems

making an automated page submission work. Our original plan was to display a page to the user that asked the user to push a form submission button. However, we found that the standard browser versions recommended for SFI's site allowed the automated form submission to the ASP, without displaying a warning to the user. This behavior is an enhancement to the integrated experience between the two sites, because to the user it appears that a page from the ASP is displayed as a result of the user's normal click on a link on an SFI page. The downside though is that the user has to wait an unusually long time for that first ASP page, because of the additional round trip through the Internet and also because of the substantial processing required at both the SFI and ASP sites.

Each arrow on the diagram represents HTTP over SSL. Therefore, the token is protected on the Internet. However, it is exposed (un-encrypted) on the user's browser, and that is the subject of the next section.

Token Crypto Operations

To protect the token while on the user's computer, there must be strong assurance that the token was not changed and could not be examined. The token must be cryptographically signed and encrypted.

We debated the use of symmetric (shared secret key) versus asymmetric[1] (public / private key pair) encryption. For a time, we favored symmetric encryption because of its simplicity and speed compared to asymmetric. When we learned more about the challenges of symmetric key management, asymmetric encryption won out. The advantage of asymmetric keys is that key management is much simpler:

1. The public key can be sent to the partner freely through non-secured medium.
2. Public keys do not have to be changed nearly as often as symmetric keys.
3. The public key can be presented in an X.509 certificate, which binds the key to the owner of the key, and contains a date to change the cert and key.
4. The X.509 certificate can be reused for SSL operations if necessary.

Our use of asymmetric encryption has proven to be a good choice over time. One ASP who has a well-defined process using symmetric keys demonstrated the complexity of symmetric key management, and I'm glad we don't have to deal with that complexity with all our ASPs.

Other technical choices were easier. The RSA asymmetric encryption algorithm was chosen because of its universality in commercial Internet applications. Based on our research, a 1024-bit key size (modulus size) appeared to be sufficient for the next 6-12 years[2] if it is changed once a year. SHA1 was generally recognized as the preferred hashing algorithm, and base64 encoding was used for text representation of binary crypto output.

There are two crypto operations applied to the token: signing and encryption.

Signing

A digital signature allows the recipient of the signed token (the ASP) to verify the sender and integrity of the message. A signature is created by hashing the token, then encrypting the hash value with the private key of the sender. The recipient may verify the signature by decrypting the hash with the sender's public key, hashing the token directly, and verifying that the two hash values match. There are additional standards available for signing (such as DSA/DSS), but for simplicity, we decided to generate our digital signatures by explicitly encrypting a SHA1 hash with the RSA algorithm using SFI's private key.

Encrypting

Because the initial 2 ASP tokens were small, they were encrypted using RSA asymmetric encryption with PKCS#1 padding[3]. This padding limits the token size to 117 bytes, but that was not

a problem with those ASPs. Later, we encountered an ASP that needed a larger token size. That was accommodated with 2 successive asymmetric encryption blocks.

The efficient way to encrypt larger data blocks (> 117 bytes in SFI's case) is to use a digital envelope: The bulk data is encrypted with a one-time symmetric key, and then the symmetric key is encrypted with the public key of the recipient. Both the bulk symmetrically encrypted data and the asymmetrically encrypted symmetric-key is sent to the recipient, who can perform the reverse operations (using the recipient's private key). However, since only one ASP ever had a larger token, we didn't develop the digital envelope capability.

We originally wanted to encrypt the token because it seemed the safe thing to do. After we worked the kinks out of the first few ASPs, we analyzed the token encryption requirement more closely. SFI's web site routinely displays confidential information to the site's users. Like any other secure site, we use technical means to keep that data as confidential as possible. We use SSL to protect the data between SFI's site and the web browser, and we use HTTP cache settings to ask the browser not to save the web page on the user's computer. Since the token exists within a browser web page, the same technical confidentiality protections apply to the token. Therefore, it is not strictly necessary to encrypt the token given that the page cache settings are correct and assuming the user is allowed to see the token data. Remember that the token is still protected against manipulation by the digital signature.

Although we're convinced the token does not need to be encrypted in theory, SFI has continued to do so. First, most ASPs have not done this analysis and just assume, like we did, that encryption is necessary. They think SFI is wrong about not needing encryption, so much so that it is easier to perform the encryption than to debate the issue. Second, although we test to be sure the digital signature is being checked at the ASP, we can't be sure that a change at the ASP hasn't stopped the signature check. Although encryption is not a substitute for a signature, it still represents a significant hurdle for an attacker, and we know the ASP *has* to perform the decryption to use the data.

Timeout at Service Provider

Normal good practice for authenticated web sites require the user to authenticate after a period of inactivity or even after a set period regardless of activity; SFI uses both types of re-authentication.

Timeouts at the ASP present a problem. At a single site, a timeout causes a re-authentication challenge screen to be displayed to user. The user then enters their credentials, the site validates them, and the user continues to use the site. This is not possible if the user times-out at the ASP. The ASP does not know the user's credentials; SFI knows them.

To remedy this, we designed a timeout flow that sends the user back to SFI, where the user enters their credentials. After authentication, the user is sent back to the ASP with a SSO token. The ASP authenticates the user based on the SFI token, and then continues to allow the user to access web pages at the ASP. The flow is shown below in Figure 2; the narrative for this figure is in the Appendix. Compare to the basic SSO flow in Figure 1.

The other way to "solve" this problem is to define it away. If the function provided by the ASP is simple, and the longest expected ASP session is less than the minimum required timeout, then the user's session can simply be destroyed. Often the ASP pages are displayed in a browser window separate from SFI's pages, in which case the additional browser window can be closed.

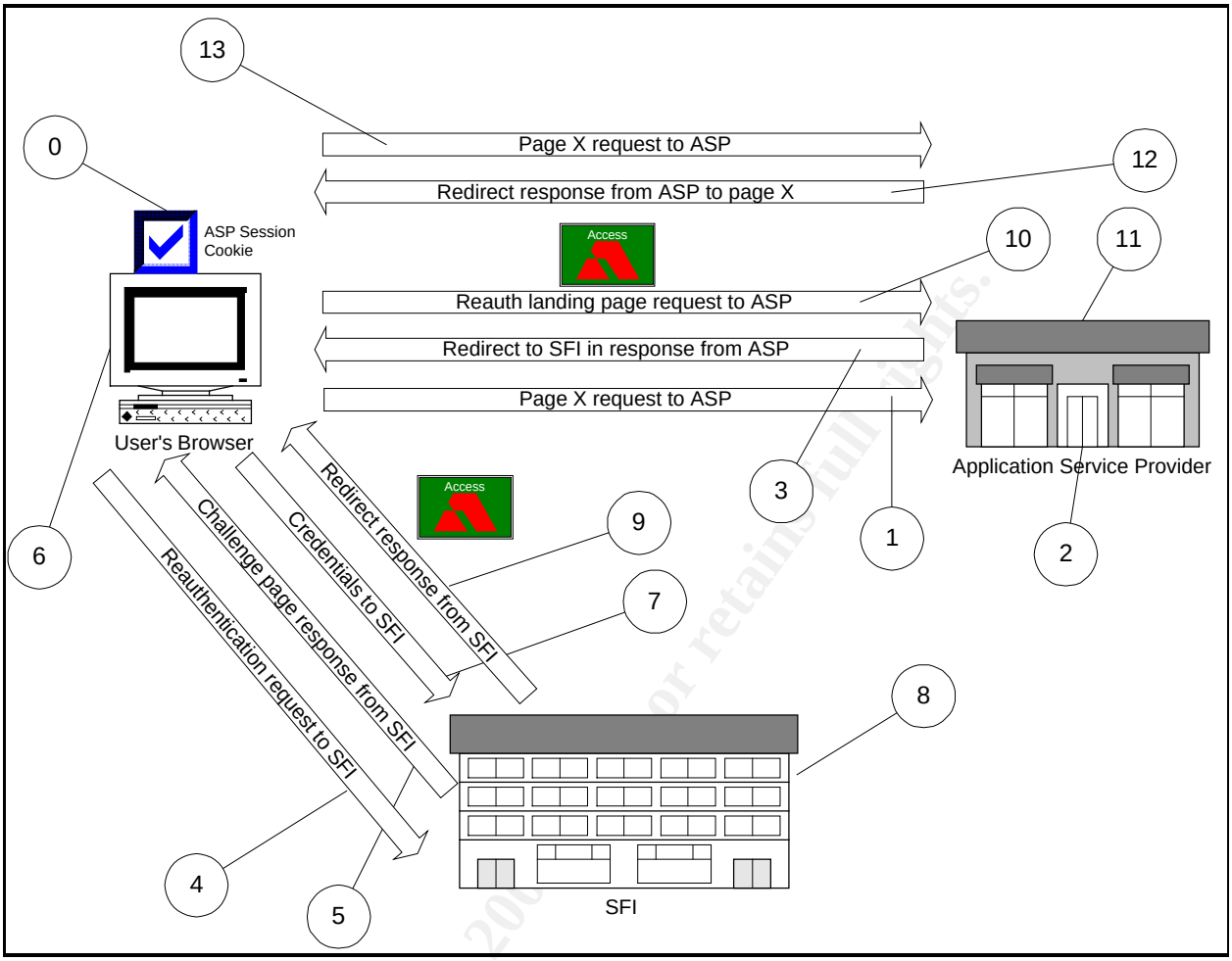

Figure 2

Back Channel Communication

The use of a token alone is often sufficient for simple services. But there are limitations for more complex services:

- There is no information flow back to SFI.
- The size of the data is limited by the maximum browser page size.
- Data in the token must make two trips over the Internet.
- All processing at both SFI and ASP adds to the user's wait time.

To overcome these limitations, a server-to-server real-time back channel communication is required. We found that the best combination of safety and simplicity was HTTP over mutually authenticated SSL. Inbound and outbound cases will be described separately.

Inbound SSL

To enable an authenticated confidential connection from ASP's server to SFI's server, XML over HTTP over SSL was chosen. The basic component on the SFI side is a standard SSL-enabled web server, so this was a relatively easy choice.

Two-way certificate-based SSL authentication was an obvious choice because of its strength, although we hadn't done certificate authentication before. Since we had to do certificate exchange anyway for the token, the marginal administrative overhead for certificate authentication was small.

In the Netscape (now iPlanet) web server that SFI used, certificate authentication is configured like other authentication methods (like "basic auth"). This specifies the set of URLs that are required to be authenticated with a certificate presented by the client end of the SSL request; in our case the ASP certificate passed in during the SSL handshake.

During the SSL handshake, the SFI web server requires the client end to present a certificate, and performs a handshake to confirm that the client side (the ASP) possesses the private key bound to the public key in the ASP certificate. The SFI web server also performs validation on the ASP certificate to confirm the certificate signature, and that the certificate was signed by a CA known to the web server (through the normal installation of CA certs in the web server).

SFI has a proprietary authentication & authorization subsystem called NSS that intercepts URL requests in the web server. To continue the authentication process started by the web server, we had to modify NSS to receive the ASP cert from the web server after the web server performs its cert checks. NSS verifies the name on the cert and matches it to the expected ASP cert. If everything checks out, the request from the ASP is allowed to proceed to the receiving application running on a Java application server.

Since authentication & authorization have already been performed by NSS at the web server, the SFI application can trust that the HTTP request came from the ASP. The application parses the XML stream, and performs the appropriate operations depending on the XML structure. The application forms an XML response back to the ASP. Depending on the nature of the request from the ASP, the response may be just an acknowledgement of data "pushed" by the ASP, or it may be SFI data "pulled" by the ASP.

The ASP Data Pull Problem

When an SFI-to-ASP connection is enabled, the possible information flows are:

1. SFI-initiated data push to ASP; enabled by Outbound SSL or token SSO flow.
2. SFI-initiated data pull from ASP; enabled by Outbound SSL.
3. ASP-initiated data push to SFI; enabled by Inbound SSL.
4. ASP-initiated data pull from SFI; enabled by Inbound SSL.

Of the four, the last one is the most risky from the viewpoint of SFI. In the other three cases, SFI controls the data to be moved. In the last case, the ASP determines what data is to be pulled from SFI.

When information is pulled from SFI by an ASP, we have to decide how much to trust the requests and how much to verify the requests. In other words, depending on how the information request is structured, it may be possible for the ASP to request more information from SFI than necessary. This could occur because of a programming mistake, or because of a dishonest person within the ASP.

We've found that some kind of compromise is usually appropriate. Some verification of the request is always done, but in most cases a completely thorough verification is unreasonably costly. The amount of verification depends on the technical possibilities in the request structure plus the amount of trust SFI has in a particular ASP. Some of the verification techniques that we implemented for ASPs include:

1. Verification of XML structure integrity. Using DTD or Schema machine validation, especially with a well-designed Schema, many logic errors and out-of-bounds requests can be caught without writing explicit logic.

2. User session association. Many ASPs have to make requests to SFI for data for a user engaged in a live SSO session. SFI validates that those types of requests only ask for data about users who have a live session open with the ASP.
3. Fine-grained authorization. The most complex option analyzes a request in detail, drilling down to make real-time authorization decisions about the nature of the data requested by the ASP.

SFI has not implemented Web Services for external partners, but the risks of data pull may become worse depending on how Web Services is implemented. There is a temptation to be overly flexible when defining standards-based communications. It is relatively easier to specify a generalized interface that returns any named data element or invokes any named class method. Therefore, SFI is on guard to carefully consider fine-grained request authorization, not just trusted partner authorization.

Outbound SSL

For processes that require SFI to initiate communication with an ASP, an outbound SSL capability was required. We needed the ability to be the "client" side of the SSL transaction, taking a role usually filled by web browsers in interactive HTTP-SSL sessions.

We bought RSA Security's *SSL-J* product, part of their *BSAFE* line. This product allows Java code to establish SSL client or server connections using Java's standard socket classes. We used Java's standard HTTP classes to initiate HTTP requests and responses.

Similar to what we needed to do on the inbound side on the web server, we used a hook available in *SSL-J* to continue certificate authentication checks after the basic SSL certificate verifications are complete. Java code compares the certificate sent by the ASP server to the ASP cert previously installed.

Example SSO Flow with Back End Server-to-Server Communication

Figure 3 shows an example SSO flow that includes a server-to-server transaction. We have supported a number of flows; some synchronous and some asynchronous with a user's session, some initiated from ASP, some initiated from SFI, some occurring without any user session. There is a variety of information exchanged with different ASPs in a variety of process flows.

In this example, once the token arrives at the ASP, the ASP requires additional information about the user. (Recall that there is a practical limit to the amount of data that can be carried in the token; this flow is one solution to that problem.) Before responding with a page to the user, the ASP must request information from SFI. In this case, the ASP creates an XML document, initiates an SSL connection with SFI, and sends an HTTP POST request containing the XML document (step 5). After certificate-based authentication of the ASP at SFI, the XML document is parsed and validated to discover what data is required by the ASP, and for which user. For this data request, SFI checks that the user represented in the XML request has (or at least could have) an active session with the ASP. Assuming all the validation checks are positive, SFI forms an XML document containing the data requested by the ASP. The XML document is returned in the HTTP response (step 6). The ASP parses the XML document, and uses that information to form the first HTML content page to the user. The page is returned to the user (step 7).

A simplified narrative of all the steps can be found in the Appendix.

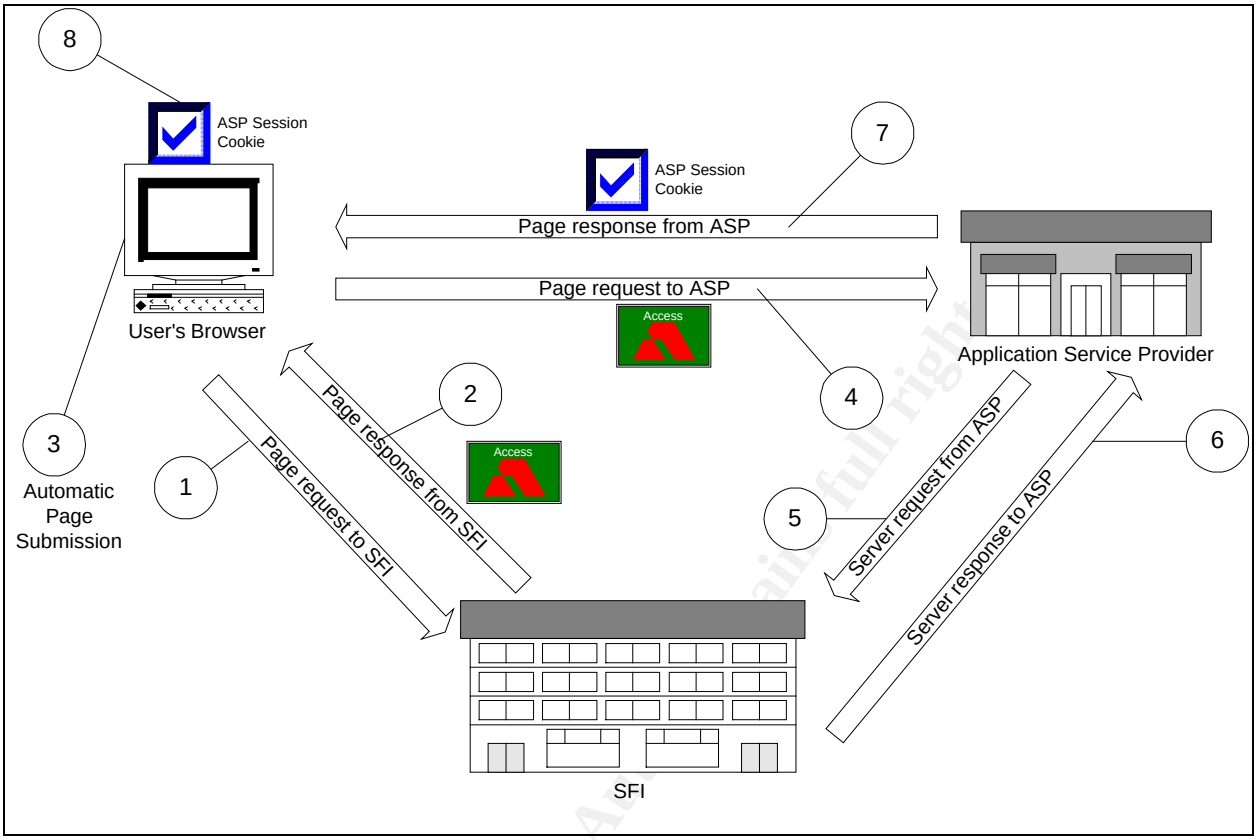

Figure 3

Infrastructure

To make the SSO flows work repeatedly, to minimize security problems and other errors, and to avoid rework, we designed and built a technical infrastructure and process controls.

Certificates and Keys

CA Trust is Not Required

The certificate signer does not have to be trusted. While PKI is based on a chain of trust to some CA, it is not necessary if your business relationships with your ASPs are one-to-one. This is certainly the case with SFI. The point of removing the trust of the CA is not that any CA is untrustworthy[4]; it is that your *technical* trust relationships should reflect your *business* trust relationships.

The implications for not trusting the CA include one-to-one certificate exchange verifications, and a check for a specific cert (not just a CN comparison) during SSL handshakes. A side benefit of not having to trust a CA is the possibility of using self-signed certs for simplicity and cost savings; although all our certs, including ASPs' certs, have been signed by CAs so far.

Certificate Management

One of our surprises when we started to use actual certificates in our applications is the necessity for cert tools. Web servers and other SSL-enabled products generally include tools for

generating and querying certs. We had to create our own basic cert tools for our custom-built cert-based applications. These included asymmetric key pair generation, certificate signing request (CSR) generation, certificate examination and validation, and a set of encryption and signing utilities for manual crypto parameter testing. The tools were written in Java, and are run from a Unix or Windows command line. All three of RSA's BSAFE Java SDKs (*Cert-J, Crypto-J, SSL-J*) were called from the command line tools to do the core standard algorithm and format work.

Certificate Exchange

Certificate exchange procedures had to be established between SFI and the various ASPs. In almost every case, the SFI cert needs to be sent to the ASP, and the ASP's cert needs to be sent to SFI. After the initial exchange, subsequent exchanges are likely to be just one-way because of different expiration dates.

We decided to use unencrypted email to send certs each way. Although confidentiality is not a concern when sending public keys, we do have to make sure an authorized employee of the organization sent the intended cert. As part of agreements between the SFI and ASP, contact lists are maintained to coordinate technical changes and fixes. The cert receiver calls the cert sender on the phone using the pre-established organizational contact list. The two parties confirm that the issuer and serial number matches, and continue to make arrangements for installation and testing.

This low-tech process of having established and tiered mutual contacts is also used to invoke emergency procedures, including certificate/key revocations and replacement if a breach is discovered. It is a faster and more reliable method than relying on a CA revocation list (CRL).

Key Management

The protection of private or[5] secret keys is the core of the security of cryptosystems. We had to develop protection mechanisms, permission standards, change procedures, and backup processes. There was also the need to design processes to safely but automatically deploy private keys to multiple application servers I can't describe any detail in this area for obvious reasons, but the reader should be aware that this a primary concern in your design.

CA Certificates

When certificates are used for authentication[6] in SSL, you need to pay attention to the installed CA certs. Since the trust model starts with your trust of a CA, you should only install CA certificates you actually trust. Even better, install only the CA certs you need in order to verify the certs of ASPs you expect. Note also the difference between consumer grades[7] versus commercial grades versus financial grades; even for CAs you trust, you only want to install the grade of cert you trust – that generally means commercial grade.

Cryptographic Services

Random Numbers

One of my biggest surprises learning about practical applications of crypto was the problem of random numbers. Random numbers are used in many crypto operations, including key generation, initialization vectors, and encryption padding. Theoretical crypto analysis starts out with the assumption that true random numbers are available. However, true randomness can only[8] be found in physical systems that exhibit measurable (Newtonian) changes based on the true randomness in quantum mechanics.

Needless to say, capturing true randomness requires unconventional hardware. Most crypto users settle for a large sample of relatively unpredictable (but not truly random) measurements.

Sometimes unpredictable events can be gathered from users interactively, but that's not practical when starting up a big application server. We settled on gathering a large sample of hard-to-predict system statistics from a large, very active computer.

Note that a random number only needs to be sampled occasionally. A pseudo-random number generator (PRNG) can create a series of hard-to-predict numbers. Although a PRNG is deterministic, the uses of pseudo random numbers are generally non-reversible. This means that the key to getting a good series of pseudo random numbers depends in large part on seeding the PRNG with a good random number[9].

Encryption and Signing Services

We used the RSA BSAFE *Crypto-J* SDK to provide the explicit encryption services needed for token signing and encryption. As more ASPs were added, we created a service layer on top of BSAFE, and required new projects to use that interface. This layer serves several purposes:

- Enforces crypto strength policy by limiting the technical availability of algorithms and parameters to only those approved for use within SFI.
- Simplifies the interface for quicker development and fewer errors.
- Integrates with SFI's key and cert infrastructure. For instance, an application only needs to use an ASP name to reference the right keys and certs for the ASP; the application does not have to be aware of path names to locate the crypto components.
- Allows SFI to replace BSAFE with a different product, if that became necessary, without effecting the many individual applications calling it.

A similar service was created for outgoing SSL to have an SFI-controlled layer above *SSL-J*. Incoming SSL does not involve an API to be called from an application, but the configuration of the URL interceptor also required just a simple ASP name to reference the cert needed for authentication.

After

Evaluation

The SSO processes and techniques we started to develop about three years ago have stood the test of time. SFI has many SSO ASPs now, and we haven't had to significantly change[10] the process since. ASPs who do not already have an established SSO readily accept our design; ASPs who already have a similar process usually adopt the changes we suggest because it strengthens their existing process.

Most of the work in the last couple of years has been to improve the formality of the ASP evaluation and development processes, and to create technical services to support the processes.

The most persistent problem we've had is certificate installation – of our cert or the ASP's cert. While we recognized early the issue of avoiding knife-edge changeover of keys and certs, we didn't consistently build a gentle changeover capability in our services nor did we emphasize the problem with our early ASPs. We also haven't mainstreamed the cert management processes as much as we would have liked by now; it's still a special process that requires the involvement of non-operational people. We're working on improving that situation, but because certs only have to be changed once a year, it's hard to generate enough urgency to complete the changes that are needed.

Other Lessons Learned

The following three items only touch on large areas that could each be the subject of another entire paper.

Service Provider Reviews

The designs described in this paper focused only on technical requirements for strong, safe, and efficient communications for single sign-on. But since information about SFI's users flows through or persists at the ASP, it is very important to review all ASPs dealing with users' sensitive information.

Describing our process of reviewing ASPs is beyond the scope of this paper. What follows are some major points to consider for your own ASP reviews, which we discovered through our experience.

- Perform a business review of the economic vitality of the ASP's business, and the business background of the ASP leadership.
- Check for security "cluefulness" on initial contact with technical representatives. One ASP suggested passing SFI's users' credentials (passwords) to the ASP to allow the users to log in at the SFI site – this indicates severe cluelessness; there would need to be extensive handholding to safely integrate that ASP.
- Perform a security review of the ASP's infrastructure and processes. We've found that on-site interactive reviews are much better than just paper reviews, and they are worth the extra cost.
- Review the proposed process (if you're not doing the design work yourself). It is important to review the big-picture information flow, and the key technical processes in extreme detail.
- Get outside help if needed. Even though SFI has gone through dozens of ASP reviews, we still employ external consultants who have done hundreds of reviews and have a much broader perspective.
- Go back and do another review when the system is actually built and ready for final testing (again, if you're not doing the design work yourself). The design often changes along the way, and seemingly small changes can have a significant impact on the technical safety or business risk in the final product.

Information Flow Trust Models

To evaluate overall business risk, SFI and its main consultants[11] created a set of standard information flow models within a risk component framework. The model framework has been a valuable tool to:

- Make fast broad judgments about the relative risk of new service provider processes.
- Influence technical or business designs to drive toward less risky processes.
- Enumerate the technical services required to support the process.

The models do not help to strengthen any single flow:

- You still have to work hard to make every individual data flow as strong as needed for the type of data in the flow.
- Each flow refers to business-level information; it doesn't matter if it's a standard Web Service or a specialized proprietary message or channel.

Testing

Doing early "unit" testing is a significant challenge. While it is important to test key processes early, especially crypto operations, it is often difficult to do so safely. If you have a reasonably secure interior network, the communications necessary for early testing affects your production network protections.

Allowing testing with your ASP will require planned changes to your environment, especially for server-to-server scenarios. Ironically, mistakes are more likely in early stages, so test planning is especially important. You also must be sure to have test keys and certs that are different from the materials to be used when the application is elevated to production.

You must be sure to verify that the ASP is actually doing the operations that you can't force them to do through technical means. For instance, the ASP has to decrypt the token, but they can get away with forgetting the vital digital signature verification. You should send bad token signatures to be sure that the ASP detects it, and you should change the client-side SSL certificate (where appropriate) to be sure the ASP denies access to your SSL request. These items should be checked in production too.

Outlook

Web Services

Web Services has been gaining industry mind share if not many real Internet implementations. The WS-Security specification is generally a superset of the specifications we developed earlier. I believe that we'll have a natural progression from our current proprietary specifications to the WS-Security specification. In particular, wider adoption of SAML will allow us to perform all the token operations we do today using higher-level commercial products instead of low-level crypto APIs. I am planning to give a talk about this subject at this year's RSA Conference.

Symmetric Encryption and Digital Envelopes

We've continued to use only asymmetric encryption, and that hasn't been a problem except in one case. Because it was only one ASP, we used an inefficient but satisfactory series of two asymmetric encryption operations. Another candidate ASP that I've already mentioned uses only symmetric encryption, and we have to use their design. SFI is also developing additional requirements around symmetric encryption of persistent data that must be handled explicitly.

Therefore, we will probably develop an explicit symmetric encryption capability along with a symmetric key management system. That system will help support the use of digital envelopes to efficiently carry more data in an SSO token.

Key Protection

While SFI has sufficient private/secret key protection today, there are hardware and software solutions available today that would improve that protection. We will probably investigate improvements to our key handling and storage, most likely in conjunction with the symmetric encryption work.

Crypto Provider Change

There is now good support for crypto operations within the application servers we use. SFI will attempt to use the standard Java crypto framework and application server crypto providers instead of the more proprietary RSA BSAFE package. We are quite happy with the BSAFE product and RSA's support, but it is much more expensive than alternatives available today that weren't available when we started.

References

Birbeck, Mark, et al. Professsional XML 2nd Edition. Birmingham UK: Wrox Press, Ltd., 2001.

Brickman, Ira. "Detailed Digital Encryption." Strategic Technology Research Treatment. 26 Jan. 2000. (Published internally in Subject Financial Institution)

Burnett, Steve, and Stephen Paine. RSA Security's Official Guide to Cryptography. New York: Osborne/McGraw-Hill, 2001.

IBM Corporation and Microsoft Corporation. "Security in a Web Services World: A Proposed Architecture and Roadmap." 7 April 2002. URL: http://www-106.ibm.com/developerworks/library/ws-secmap/ (26 Mar. 2003).

IBM Corporation and Microsoft Corporation. "Web Services Security (WS-Security)." 5 April 2002. URL: http://www-106.ibm.com/developerworks/library/ws-secure/ (26 Mar. 2003).

RSA Laboratories. "Public Key Cryptography Standards." URL: http://www.rsasecurity.com/rsalabs/pkcs/ (26 Mar. 2003).

RSA Laboratories. "RSA Laboratories' Frequently Asked Questions About Today's Cryptography." 4.1. URL: http://www.rsasecurity.com/rsalabs/faq/sections.html (26 Mar. 2003).

Srivastava, Ajay, and Dino Battaglia. "Digital Signatures." Strategic Technology Research Treatment. 17 Dec. 1999. (Published internally in Subject Financial Institution)

Glossary

This list contains definitions appropriate to the body of this paper. You may find more accurate or more accepted definitions from my References.

ASP	Application Service Provider. Although *ASP* can refer to broader definitions, in this paper an ASP offers a web-based service to individuals on behalf of SFI.
Base64	An Internet text encoding standard to convert binary data into characters from the set { a-z, A-Z, 0-9, /, + }.
BSAFE	A family of crypto SDKs from *RSA Security*.
CA	Certificate Authority. An organization that signs public key certificates.
Cert	Public key certificate. See also PKI.
CRL	Certificate Revocation List. A cert owner or cert signer has to be able to revoke a cert if a problem occurs, such as a compromise of the private key. A CRL is the publication of those revocations, generally machine readable to verify a cert. SFI relies on one-to-one revocation procedures instead.
Crypto	Cryptographic or Cryptography, depending on adjective versus noun context.
CSR	Certificate Signing Request. A standard format for information to be sent from a certificate requestor to a certificate signer (a CA). It contains the public key to be signed, the name of the owner, and a signature formed from the private key of the pair.
Hash	A reproducible irreversible number representing the combination of a stream of data, or the operation to produce such a number. Cryptographically strong hash functions include SHA1 and MD5.
OFX	Open Financial Exchange. A protocol for exchanging information between financial institutions or software. A special OFX cert is needed to enable OFX transactions over the Internet; this cert is only available to qualified financial institutions.
PKCS	Public-Key Cryptography Standard. Individual standards look like "PKCS #1", which is the standard for the RSA algorithm. See the reference section for a link to the standards.
PKI	Public Key Infrastructure. In general, a system for validating public keys used for asymmetric (public/private) encryption. Usually infers X.509 standard certificates, certificate authorities (CAs) that sign such certificates, and RSA algorithm keys. Also can refer more broadly to the components and users of that system.
PRNG	Pseudo Random Number Generator. Given a seed, a PRNG produces a series of hard-to-predict numbers. A good (random) seed is vital to the strength of the PRNG output.
Proxy	As used in this paper, it is the SSO technique that forces all ASP requests through SFI, to allow SFI to apply user identity data before the ASP receives the request.
RSA	A strong and commercially popular asymmetric (public/private key pair) encryption algorithm.
RSA	Short for *RSA Security*, a major vendor of technical security products.

Salt	A number combined with the subject data to strengthen a hash. It is generally used to reduce the chance of table-matching hashes of small data streams.
SDK	Software Development Kit. A software package used by programmers to access pre-written functions from their program code.
SFI	Subject Financial Institution. The name used in this paper to refer to the specific financial institution for which the author performed the work described in this paper.
SHA1	A strong hashing algorithm taking any stream of data as input to produce a 160-bit number as output.
SSL	Secure Sockets Layer. A widely used standard for confidential and authenticated communication over a TCP/IP connection.
SSN	Social Security Number. An SSN is issued to individuals, while an Employee Identification Number (EIN) is issued to organizations.
SSO	Single Sign-On. AKA Consolidated Logon. The ability for a user to sign-on just once to a web site, and authenticate to another web site without the user having to present credentials again. In this paper the first web site is almost always the SFI site.
TLA	Three-Letter Acronym.
Token	As used in this paper, it is the encapsulated user identity data passed from SFI through the user's browser to the ASP. It also describes the SSO technique that uses a token, in contrast to the *proxy* method.
X.509	The most popular public key certificate standard.

Appendix

Narrative for Figure 1

0. (not shown) User authenticates to SFI and establishes session with SFI
1. User clicks on link to initiate service, causing page request to SFI.
2. SFI creates an SSO token containing the identity of the user as understood by the service provider.
3. The token is sent back to the browser in a "hidden" form element on a page containing form submission code.
4. The user's browser automatically submits the form containing the token to the service provider.
5. A page request is sent to the service provider including the token.
6. The service provider decrypts the token, verifies the timestamp, verifies SFI's signature, and creates a session for the user.
7. A cookie with a session id is sent back to the user's browser along with the first page of the service. The page contains links back to the service provider.
8. The service provider's session cookie is stored in the user's browser for subsequent page requests to the service provider.

Narrative for Figure 2

0. (not shown) Initial SSO has already been accomplished, and the user is navigating around the service provider site.
1. User makes a request for some service provider page named X.
2. The service provider session times-out (could be hard timeout or activity timeout).
3. Service provider responds to request for page X with a redirect command to a special SFI re-authentication URL.
4. Request from browser to SFI re-authentication URL.
5. SFI invalidates its session and responds with authentication form.
6. User enters their credentials and submits the form.
7. Form data credentials submitted to SFI.
8. SFI validates credential, refreshes its session, and creates authentication token (similar to original SSO token).
9. Response to form submission includes token and automatic submission script to special service provider URL.
10. Request to service provider's re-authentication landing page including submission of token from SFI.
11. Service provider validates SFI token, refreshes its session, and determines original page request X from step 1.
12. Service provider sends a redirect to page X.
13. Original page X request is sent from browser to service provider.

Narrative for Figure 3

0. (not shown) User authenticates to SFI and establishes session with SFI
1. User clicks on link to initiate service, causing page request to SFI.
2. SFI creates an SSO token containing the identity of the user as understood by the service provider. The token is sent back to the browser in a "hidden" form element on a page containing form submission code.
3. The user's browser automatically submits the form containing the token to the service provider.
4. A page request is sent to the service provider including the token. The service provider decrypts the token, verifies the timestamp, verifies their signature, and associates the existing session for the user.
5. The service provider requests information about the user from SFI.
6. SFI returns the information requested back to the service provider.
7. A cookie with the session id is sent back to the user's browser along with the first page of the service. The page contains links back to the service provider.
8. The service provider's session cookie is stored in the user's browser for subsequent page requests to the service provider.

Endnotes

[1] This paper is not meant to be a primer on applied cryptography. However, a brief description of asymmetric encryption may be helpful. The reader is encouraged to use the references provided for more details.

Asymmetric encryption uses two keys, a public key and a private key. They are generated from a random number in such a way that one cannot be derived from the other. Data that is encrypted with a public key can *only* be decrypted with the private key from the pair. The opposite is also true; data that is encrypted with a private key can only be decrypted with the public key from the pair. The owner keeps the private key secret. The public key is given freely; it is not a secret. This solves the difficult problem of having to exchange keys over a secure channel.

A public key certificate binds the key to the key's owner. A *certificate authority* digitally signs a certificate. The certificate authority vouches that the name on the certificate legitimately owns the enclosed public key. To trust the name on a certificate, a person must trust the authority that signed the certificate, or trust one certificate authority in a chain of certificate authorities.

[2] The strength of encryption relies on computational infeasibility: the computational power and time needed for keyless decryption exceeds any reasonable time and cost bounds. As computational power per dollar increases, the infeasibility becomes less and less. Therefore, effective key strength decreases as time goes on.

[3] Padding is necessary because the native RSA algorithm encrypts data blocks the same size as the key – 128 bytes for the SFI chosen key size. If a block of data is less than the key size, extra bytes have to be added to avoid weakening the encryption while allowing the decryption to recognize the true size of the original data. The padding standard we chose, PKCS #1 "block 02", reduces the maximum data block size to 117 bytes.

[4] Although VeriSign has had a couple publicized problems.

[5] *Private* keys are used in asymmetric encryption, *secret* keys in symmetric encryption, but both kinds are "secret" in the sense that they must be highly protected against compromise – loss, change, and especially capture by unauthorized persons.

[6] Especially if you're relying on a CA's signature to vouch for an incoming certificates; SFI does not rely on the CA's signature alone, but it's still a first line defense.

[7] CAs require different levels of proof of identity for different levels of certificate. For instance, VeriSign's lowest level only needs confirmation of an email address. Their highest commercial grade requires extensive proof of ownership of a name. Financial grades include specialized OFX certs and "global" certificates that allow a browser to step up encryption levels.

[8] I never actually read this, but every good example I found in the literature seemed to get its randomness from quantum mechanical randomness – electromagnetic noise, semiconductor noise, radioactive decay.

[9] When I first read about the requirement for a pseudo-random number generator needing a good random number, I was reminded of a certain comedian's joke (paraphrasing): "I can tell you how to make two million dollars – guaranteed. First, get a million dollars. Then …"

[10] There is one candidate ASP who has a complex but well-designed SSO process that uses symmetric keys. They were mentioned earlier in passing to make the point that the complexity of that solution further reinforces the wisdom of using asymmetric keys.

[11] SystemExperts; www.systemexperts.com

Last Updated: January 21st, 2019

Upcoming Training

SANS Las Vegas 2019	Las Vegas, NV	Jan 28, 2019 - Feb 02, 2019	Live Event
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
Security East 2019 - SEC401: Security Essentials Bootcamp Style	New Orleans, LA	Feb 04, 2019 - Feb 09, 2019	vLive
SANS Anaheim 2019	Anaheim, CA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Northern VA Spring- Tysons 2019	Tysons, VA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS New York Metro Winter 2019	Jersey City, NJ	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Dallas 2019	Dallas, TX	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Secure Japan 2019	Tokyo, Japan	Feb 18, 2019 - Mar 02, 2019	Live Event
SANS Scottsdale 2019	Scottsdale, AZ	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Reno Tahoe 2019	Reno, NV	Feb 25, 2019 - Mar 02, 2019	Live Event
Open-Source Intelligence Summit & Training 2019	Alexandria, VA	Feb 25, 2019 - Mar 03, 2019	Live Event
Mentor Session @Work - SEC401	Raleigh, NC	Feb 27, 2019 - Mar 06, 2019	Mentor
SANS Baltimore Spring 2019	Baltimore, MD	Mar 02, 2019 - Mar 09, 2019	Live Event
Baltimore Spring 2019 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Mar 04, 2019 - Mar 09, 2019	vLive
Community SANS Indianapolis SEC401	Indianapolis, IN	Mar 04, 2019 - Mar 09, 2019	Community SANS
SANS Secure India 2019	Bangalore, India	Mar 04, 2019 - Mar 09, 2019	Live Event
SANS Secure Singapore 2019	Singapore, Singapore	Mar 11, 2019 - Mar 23, 2019	Live Event
SANS London March 2019	London, United Kingdom	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS San Francisco Spring 2019	San Francisco, CA	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS St. Louis 2019	St. Louis, MO	Mar 11, 2019 - Mar 16, 2019	Live Event
Mentor Session - SEC401	Fredericksburg, VA	Mar 12, 2019 - May 14, 2019	Mentor
SANS Secure Canberra 2019	Canberra, Australia	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS Norfolk 2019	Norfolk, VA	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS Munich March 2019	Munich, Germany	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS vLive - SEC401: Security Essentials Bootcamp Style	SEC401 - 201903,	Mar 19, 2019 - Apr 25, 2019	vLive
SANS 2019 - SEC401: Security Essentials Bootcamp Style	Orlando, FL	Apr 01, 2019 - Apr 06, 2019	vLive
SANS 2019	Orlando, FL	Apr 01, 2019 - Apr 08, 2019	Live Event
Community SANS Raleigh SEC401	Raleigh, NC	Apr 01, 2019 - Apr 06, 2019	Community SANS
SANS London April 2019	London, United Kingdom	Apr 08, 2019 - Apr 13, 2019	Live Event
Blue Team Summit & Training 2019	Louisville, KY	Apr 11, 2019 - Apr 18, 2019	Live Event
SANS Riyadh April 2019	Riyadh, Kingdom Of Saudi Arabia	Apr 13, 2019 - Apr 18, 2019	Live Event