

---

# HIPAA in the Cloud

## Technical Architectures that Render PHI as 'Secured'<sup>i</sup>

*Author: Mick Seals, Senior Manager Consultant  
Sogeti USA, Inc.*

*Version 1.2*

*Published: October, 2011*

---

### **Abstract**

This paper introduces architectures that can be used to secure HIPAA (Electronic) Protected Health Information. While specifically addressing the encryption and decryption of data at rest and in-transit in the Cloud, this information applies equally well to protecting information residing on-premise.

The techniques assure that data at rest and data in motion anywhere within a solution is encrypted until it is required for processing (data in use).

Other aspects of the HIPAA requirement regarding auditing, training and policy are organizational and beyond the scope of this paper.

*This whitepaper is for informational purposes only.*

*This whitepaper is not intended to constitute legal advice. You are encouraged to seek the advice of counsel regarding compliance with HIPAA and other laws that may be applicable to you. Sogeti USA LLC and its affiliated entities make no representations or warranties that following the advice presented here will assure compliance with applicable laws, including but not limited to HIPAA.*

*THIS WHITE PAPER IS PROVIDED "AS IS" AND WITHOUT ANY WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED. WITHOUT LIMITATION, THERE IS NO WARRANTY OF NON-INFRINGEMENT, NO WARRANTY OF MERCHANTABILITY, AND NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE. ALL WARRANTIES ARE EXPRESSLY DISCLAIMED.*

*USER ASSUMES THE FULL RISK OF USING THIS SPECIFICATION. IN NO EVENT SHALL SOGETI USA BE LIABLE FOR ANY ACTUAL, DIRECT, INDIRECT, PUNITIVE, OR CONSEQUENTIAL DAMAGES ARISING FROM SUCH USE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

*Microsoft, Windows Azure, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.*

*© Copyright Sogeti USA LLC 2010. Sogeti® is a registered trademark of Sogeti, USA LLC or its affiliates in the United States or other countries. All other trademarks are the property of their respective owners.*



---

## Contents

<b>Contents</b> .....	<b>3</b>
<b>Figures</b> .....	<b>5</b>
<b>Introduction to HIPAA and Electronic Protected Health Information</b> .....	<b>1</b>
<b>HIPAA Technical Safeguard Requirements</b> .....	<b>3</b>
Access Control .....	4
Unique User Identification .....	5
Emergency Access Procedure .....	5
Automatic Logoff .....	5
Encryption and Decryption .....	5
Audit Controls .....	9
Integrity Controls .....	9
Person or Entity Authentication .....	10
Transmission Security .....	10
Integrity Controls .....	10
Encryption .....	10
<b>Additional HHS Guidance</b> .....	<b>11</b>
Disaster Recovery .....	11
Database Migration vs. Synchronization vs. Backup .....	12
<b>Additional Guidance</b> .....	<b>14</b>
Implement a Key Store .....	14
On-Premise Key Store .....	14
In-the-Cloud Key Store .....	15
Encrypt Configuration Files .....	16
Encrypt Output Files to Hidden Azure Blobs, then Expose them via an HTTPHandler .....	16
Encrypt Working Files .....	16
Satisfying the Business Associates Agreement Requirement .....	17
Encrypting Initial ETL Data Load .....	18
Thin-Client Cryptography .....	18
RIA-Client Cryptography .....	19
Searching on Encrypted Data .....	19

Reporting on Encrypted Data ..... 19

**HIPAA Technical Safeguards ..... 21**

**Summary ..... 22**

**About the Author ..... 23**

    About Sogeti USA, LLC ..... 23

**Related Links ..... 24**

---

## Figures

Figure 1 - Reference Azure Architecture .....	4
Figure 2 - The SQL Azure Firewall Settings Dialog.....	7
Figure 3 - Typical Code and Data Migration with Production SQL Azure Failover .....	12
Figure 4 - On-Premise Key Store .....	15
Figure 5 - Azure Table Storage Key Store .....	15

---

## Introduction to HIPAA and Electronic Protected Health Information

In 1996 congress enacted the **Health Insurance Portability and Accountability Act (HIPAA) of 1996 (P.L.104-191) [HIPAA]**<sup>1</sup>. This act provides for standards for protecting Protected Health Information (PHI)<sup>ii</sup> by Covered Entities (organizations required to protect PHI entrusted to them). The law now stipulates that sub-contractors must meet the same security requirements as the Covered Entity, so for our purposes as technology consultants we must treat any electronic records system we design to be protected by the same standard of care as applies to the Covered Entities themselves.

HIPAA includes reporting requirements by Covered Entities for any known breach of PHI within 60 days of its discovery. As of April 4, 2011, OCR reported that a total of 256 breaches have impacted 10,202,051 persons in breaches reported by Covered Entities from September 22, 2009. A breach is not only embarrassing, it can be costly. The Covered Entity must publicly post the breach and notify all affected parties. The affected parties may also have access to civil recourse for any damages caused by the breach.

Interestingly, if data is encrypted using one of the proscribed technologies, a system intrusion and resulting disclosure of an encrypted piece of information is not considered to be a breach as defined for the purposes of this reporting requirement.

The HIPAA law itself provides only that standards are to be established by the government, and as is often the case, the actual rule-making effort is undertaken by a Federal agency. In this case that agency is the Department of Health and Human Services.

HHS has implemented guidance documents for the following areas of the law:

- Administrative Safeguards (Process, Audit and Authorization of Users)
- Physical Safeguards
- Technical Safeguards

Administrative safeguards are those related to the organization determining who should be authorized to see PHI, an audit of who has accessed PHI and the organizational policies and procedures in place to assure that when a person or system no longer needs access to PHI, that such access is revoked.

This part of the rule contains language about audits, policies, procedures, contingency plans and disaster recovery.

Microsoft has addressed many of these administrative policies and physical safeguards that address specific HIPAA requirements in its Microsoft Compliance Framework<sup>2</sup>.

While these are important topics, they do not have special meaning when a system is implemented in the cloud, and so are outside the scope of this whitepaper.

---

<sup>1</sup> [http://en.wikipedia.org/wiki/Health\\_Insurance\\_Portability\\_and\\_Accountability\\_Act#Security\\_Rule](http://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Accountability_Act#Security_Rule) For general information on HIPAA.

<sup>2</sup> <http://www.globalfoundationservices.com/documents/MicrosoftComplianceFramework1009.pdf> - Microsoft Compliance Framework.

Likewise Physical Safeguards are important for HIPAA compliance, but the physical safeguards of most cloud datacenters are either well-documented or are included in other existing datacenter certification standards.

Of course, even if a datacenter is protected against being physically compromised, certainly an application can be built in such a way that PHI is exposed to the public. Especially when a system is deployed to the cloud, it should be designed so that PHI is protected even in the case of a security breach.

A note of caution regarding HIPAA safeguard requirements. There are certain requirements that are advised, and others that are 'exhaustive'. When a security safeguard is exhaustive it means that only those specific technologies listed are acceptable. These will be identified when they apply to an architecture element.

Many of the safeguard requirements are broader and open to interpretation. In these cases you should use your best judgment and default to the more strict interpretation where possible.

When HIPAA was first enacted business partners could hide behind the client's own responsibility statements. This was rectified in a follow-on law called HITECH which requires that business partners that handle unprotected PHI sign a Business Associates Agreement with the Covered Entity. However, a Covered Entity *may* not be required to sign this agreement if the following conditions are met:

- The PHI is encrypted outside of the cloud platform.
- The encryption keys are stored in an on-premise key store.
- Unencrypted PHI is never stored or handled in the cloud.

This paper details strategies that meet these qualifications to remove the Business Associates Agreement requirement by encrypting data before bulk load into SQL in the cloud, and encryption and decryption of data at the client level in the browser or client application itself.

Another note in regard to 'HIPAA Audits' and 'HIPAA Certifications'; the HHS does not certify auditors to conduct HIPAA compliance certifications.<sup>3</sup> HHS also does not certify datacenters or applications. Instead, it has issued guidance documents specifying what it considers to be necessary in order for an organization to be in compliance with the law.

You should work with your security consultant and with the security officer of the applicable Covered Entity in establishing your own security strategy.

The techniques discussed in this whitepaper can help you to implement technical safeguards as part of that overall strategy.

Keep in mind that they can also be used in your on-premise systems to provide a higher level of security and HIPAA compliance.

---

<sup>3</sup> <http://www.hhs.gov/ocr/privacy/hipaa/understanding/coveredentities/misleadingmarketing.html>

---

## HIPAA Technical Safeguard Requirements

When Congress passed the HIPAA law (*The Law*), it deferred implementation of specific technical requirements to the department of Health and Human Services (HHS) (*The Rules*). HHS has issued guidance regarding steps that can be taken that indicate a good faith effort has been made to comply with the HIPAA law.<sup>4</sup> HHS, not being experts in technical data protection standards, deferred to existing data protection guidance implemented by the National Institute of Standards and Technology (NIST). (*The Guidance*)

For all the discussion and concern regarding *The Rules*, they are remarkably straight forward, encompassing a total of 5 standards: (The entire Rule is included in the Appendix)

- Access control
- Audit controls
- Integrity
- Person or entity authentication
- Transmission security

Most standards include Implementation Specs, each of which can be either Required or Addressable. For our purposes the distinction is meaningless. We must treat any Addressable rule as being a Required rule.

I will be using the following reference architecture when discussing the various security components.

---

<sup>4</sup> <http://www.hhs.gov/ocr/privacy/hipaa/understanding/srsummary.html>

Azure Architecture Overview  
Version 1.0

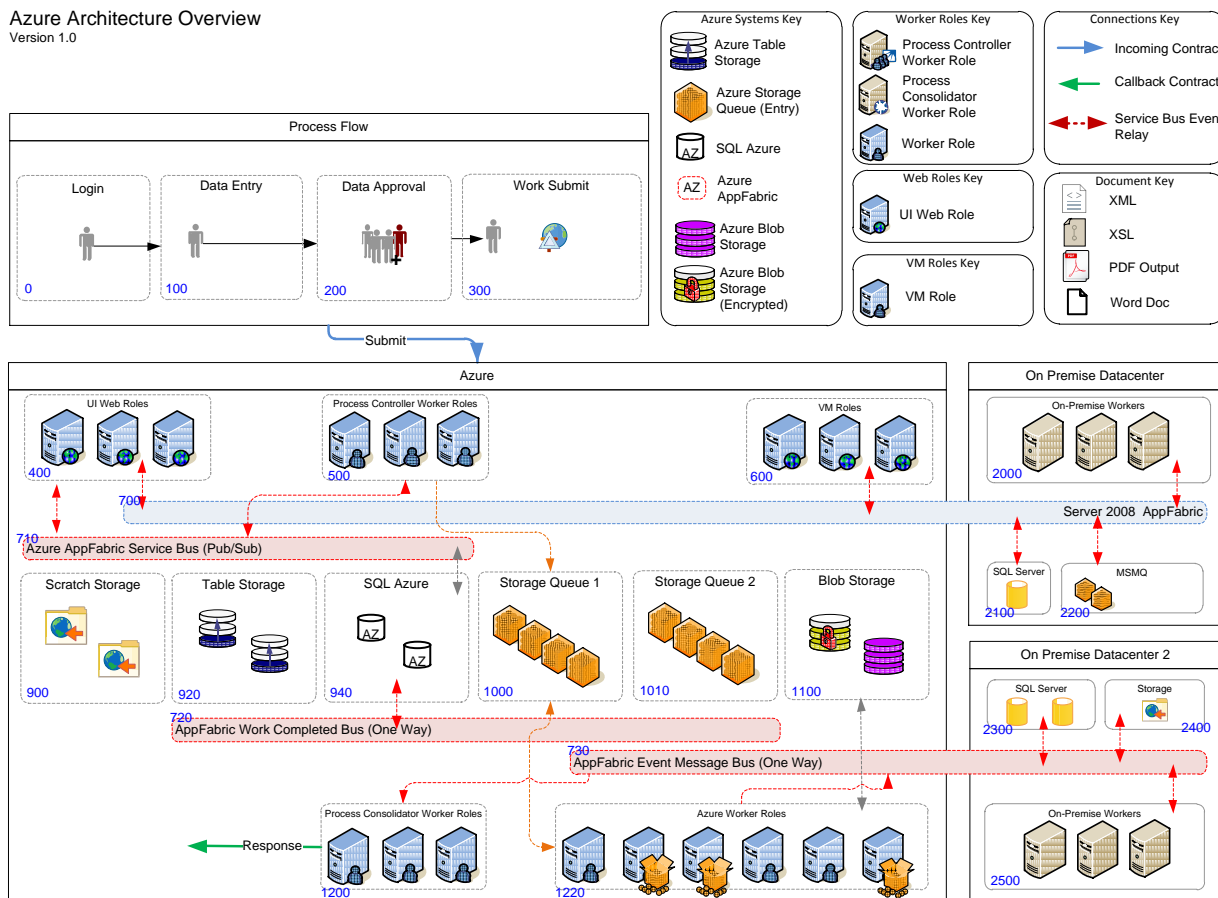


Figure 1 - Reference Azure Architecture

### Access Control

Access Control is the process of assuring that only authorized persons or processes have access to the PHI in a system. This includes the concept of user authorization but also the integrity of the sign-in process and preventing unauthorized parties from logging into a system.

You should extend this definition to include authorization and encryption of intra-solution processes running on Web<sup>400</sup>, Worker<sup>500</sup> or VM<sup>600</sup> Roles.

The Rule includes the following Specifications:

- i. Unique user identification (Required) Assign a unique name and/or number for identifying and tracking user identity.
- ii. Emergency access procedure (Required). Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency.
- iii. Automatic logoff (Addressable). Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity.
- iv. Encryption and decryption (Addressable). Implement a mechanism to encrypt and decrypt electronic protected health information.

## Unique User Identification

Don't allow your system users to share usernames and passwords. Each individual<sup>100</sup> must be assigned his own account, with all access to the system logged. The system should also maintain a log of each record that the defined user accessed and when, along with an audit trail of all Reads, Inserts, Updates and Deletes. There should be safeguards in place to revoke access to the system once it is not needed.

## Emergency Access Procedure

### Recommendation:

Implement SQL Server replication or otherwise copy the Cloud database to an on-premise database or alternate Cloud for backup and emergency access.

If your Web site is connecting to a database in the cloud<sup>2100</sup>, this Spec requires that you have alternate access to the same data through an alternate mechanism in the case of an emergency. You may want to consider implementing a database replication strategy and an alternate, backup Web site that provides for emergency access to it. This would not need to have access to the full functionality of your main Web site, just access to data if needed in an emergency. Technologies available include SQL Server replication (SQL Azure can be a source or target for an on-premise instance of SQL Server running the replication), SQL Azure Sync or the open source tool SQL Azure Migration Wizard.<sup>5</sup> The SQL Server Migration Wizard can be operated from the command-line on a scheduled basis.

## Automatic Logoff

If you are using the standard ASP.Net membership provider (you should be), then you can simply configure it to automatically log the user out after a pre-determined period of time.<sup>6</sup> This is traditionally set to 20 minutes. It also means that you cannot offer your users the 'Keep me logged in' option.

## Encryption and Decryption

The Encryption and Decryption spec specifically regards the encryption and decryption of data-in-motion and data-at-rest. At some point data needs to be processed in an unencrypted form (data-in-use). One goal of a highly secure system is to reduce the surface of attack by keeping unencrypted data in memory only shortly. If you have a choice between storing PHI in a static variable vs. a transient variable, choose the latter. It should never be stored for any length of time in the session state or view state.

Consider the use of SecureString as opposed to String or StringBuilder in your Dot.Net code. SecureString provides the following automatic safeguards:

- Encryption (protects disclosure in memory dumps or page caching.)
- Pinning in memory (prevents the garbage collector from making extraneous copies.)
- Ability to mark as read-only.
- Construction is allowed by appending single characters only.

There are two elements to the Encryption Spec. The first is Identity Confirmation and the second is transmission security.

---

<sup>5</sup> See <http://sqlazuremw.codeplex.com/>

<sup>6</sup> See <http://support.microsoft.com/kb/315584>

### Identity Confirmation

**Recommendations:**

Use Extended Validation certificates instead of standard certificates on public-facing Web sites that run in the cloud.

Use self-signed X.509 certificates for intra-role communication, authentication and encryption (SSL).

A user of a system that contains PHI must be assured that he is actually hitting the system he thinks he is. If a hacker were able to impersonate the login page of a site, then the user may inadvertently provide his username and password to a third party.

In regards to Identity Confirmation, this standard does not have special meaning when a system is implemented in the cloud vs. an on-premise system. Imagine a system that implements the architecture represented in Figure 3. A traditional Web UI built in ASP.Net allows a person to log in through a Web interface<sup>400</sup>. This system must assure that the login is secure. This is traditionally accomplished just as in an on-premise system through the use of certificates signed by a Certificate Authority such as Verisign.

The basic certificate that can be installed on an IIS server (and uploaded to the Cloud as an X.509 certificate) depends on a validation that the email address of the requester is the same domain as that of the site being protected, simply because it is unlikely that someone from outside an organization could 'spoof' his email address when making the request.

This is a base level of security that probably rises to the level of due diligence required by the rule. If this base level of certificate is used, the risk that an invalid certificate could be issued to a third party must be documented in the risk assessment (which is a required aspect of a system being HIPAA compliant.)

**Recommendation:**

Implement Certificate Authority-signed SSL certificates on the Web UI Role to enable HTTPS communications.

Implement X.509 self-signed certificates for intra-role communications.

Extended Validation certificates provide a higher level of assurance to the user. EV certificates go through additional identity validation steps by the issuing authority, so their presence should present an additional assurance that when a person enters their username and password, they are not accidentally providing it to a third party.

### Transmission Security

Transmission Security applies to PHI that is transmitted between the database and application servers, between application servers<sup>700-740</sup>, and between the Web server and the user<sup>100-400</sup>. This is referred to as 'Data-in-Motion' in the HHS guidance.

Data moving between the Web server and the user can easily be secured using an SSL certificate signed by a Certificate Authority (like Verisign). These certificates are uploaded into the Azure Management Portal and serve to validate the identity of the server on the other end and also to act as the encryption key for Secure Sockets Layer (SSL) traffic between the Web server and the user's browser itself.

We recommend 128 bit SSL certificates for a high level of security. As machines become more powerful with the prospect of brute-force cracking of the security key, this level of protection should be increased over time.

Communication between the application or Web server and an on-premise database<sup>400-2100</sup> should be set to use the SSL connection mechanism.<sup>7</sup> (SQL Azure requires it.)

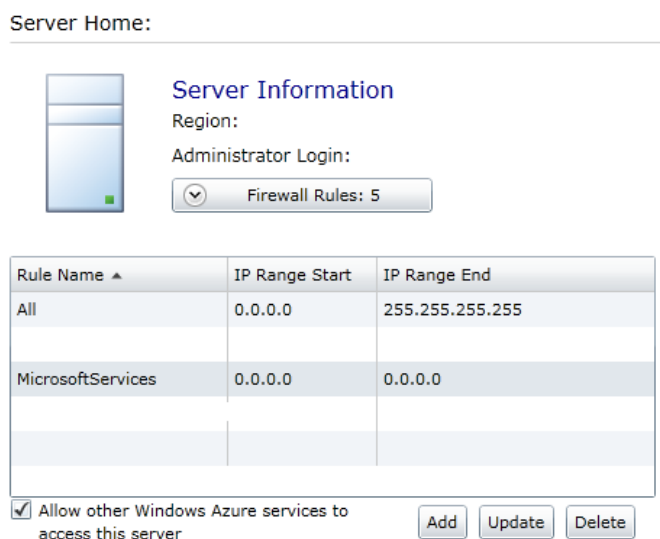
Set `Encrypt=True` and `TrustServerCertificate=False` in the database connection string.

### Application Servers

Many Cloud solutions will include more than just a public-facing Web Server; they may also include application servers<sup>500,600,2000</sup> that need access to databases and may be wired together using Web Services. While Azure in particular wraps all servers in a subscription inside of a firewall-protected 'envelope', all servers are placed in the same domain.

### Role-to-SQL Azure Communication

SQL Azure is an exception, as it resides in its own domain within Azure and is granted access to your subscription roles by selecting a checkbox on the Azure Management Portal:



**Figure 2 - The SQL Azure Firewall Settings Dialog**

Information flowing between your solution and SQL Azure, even if hosted within the same subscription, can be considered to be travelling across the open Internet. Store the connection string in a secured key store or encrypt the configuration file.

### Intra-Role Communication

Many applications are SOA-enabled, with the functionality of the system broken down into Web Services that call each other. Web Services are arranged into logical units that may run on separate Web<sup>400</sup> or Worker<sup>500</sup> Roles.

In an on-premise datacenter the machines that host these application services are protected *behind a second firewall* that is not present in Azure. In a traditional configuration, only the UI Web server is exposed directly to the public. Then holes are punched in the second firewall to grant access from this specific Web server to the specific IP addresses of the application and database servers it needs. The

<sup>7</sup> <http://msdn.microsoft.com/en-us/library/ff394108.aspx> - Security Considerations of SQL Azure.

application servers and database server are free to communicate amongst themselves freely and without encryption or authentication because they are safely behind two firewalls.

Application servers in Azure, whether implemented as Web, VM or Worker roles, must be treated as if they are public-facing machines sitting in a DMZ behind a single firewall.

Technically, they are behind one physical and one logical firewall. Azure wraps all servers in a subscription into a separate network segment, walled off by a physical firewall. Then each machine itself has Windows Firewall turned on, defaulting to all ports closed unless explicitly opened in the configuration files. Ports 80 and 443 are open by default on Web Roles. Ports can be opened for internal machines, external machines or both.

### *Transmission Security*

The EV certificate that provides Identity Confirmation also provides transmission security by providing the encryption key to the Secure Sockets Layer (SSL). In a Web browser this transport is indicated by HTTPS:// in the URL.

Technically, transmission security also applies within a Cloud computing environment to the data that is passed back and forth within a computing environment. Just as the Certificate Authority-signed certificate on a Web server validates that you are communicating with the intended server (EV) and that the packets sent back and forth have not been intercepted by a 'man-in-the-middle' (via SSL), we need the same levels of assurance with our intra-application communications. These channels can be protected with self-signed X.509 certificates.

You should be building your applications on an SOA architecture to break your application down into reusable service components but also so that intra-application traffic can be secured. When you use Web services, you can use standard Internet protection mechanisms like SSL. This applies equally well to Web Services and to REST interfaces. Both are built on the HTTP(s) stack and leverage the security mechanisms of the Web.

SSL can also be used with Named Pipes.<sup>8</sup>

### *Encrypting SQL Azure Data-at-Rest*

In an on-premise instance of SQL Server 2008 EE<sup>2100</sup>, you can set up your server so that the native database file itself is encrypted at the operating system level.<sup>9</sup>

SQL Azure<sup>940</sup> does not implement this feature at either the native file level (the storage subsystem of SQL Azure is hidden from us), or at the cell level (CLE).

Certainly data in a database is considered to be data-at-rest and subject to the HIPAA protections. So until Microsoft implements native database encryption we are left with the need to implement encryption at the application level.

This can have a significant impact on performance (perhaps 20%) and on the discoverability of data (search).

---

<sup>8</sup> <http://blogs.msdn.com/b/mwasham/archive/2011/03/30/migrating-a-windows-service-to-windows-azure.aspx> - Security on Named Pipes in Azure

<sup>9</sup> [http://msdn.microsoft.com/en-us/library/cc278098\(v=sql.100\).aspx](http://msdn.microsoft.com/en-us/library/cc278098(v=sql.100).aspx) – Native encryption in SQL Server 2008 EE

This mechanism can be implemented at the application layer if your app has a middle tier. You probably already have implemented some form of entity framework in a middle tier that maps database operations to entity objects that allow you to add data access layer functionality of just this kind.

Most people do not write these layers by hand. The Entity Framework services will build entire CRUD operations against complex databases. Some systems implement separate Data Access Layers and Business Logic Layers. The appropriate location for an encryption/decryption mechanism would be at the DAL or the BLL if no DAL exists.

**Recommendation:**

Use hash columns to validate that data and files have not been altered.

While the details of the exact mechanism for encryption at the application layer are beyond the scope of this whitepaper, it is important to keep in mind that the HIPAA guidance proscribes specific encryption technologies that may be used. You cannot use a hash (like MD5 or SHA1) as these are not on the list. OpenPGP is acceptable because the default encryption method is AES, which is.

## Audit Controls

Your solution must keep track of all system and user accesses to PHI data. This audit trail must be monitored by a human in order to validate that there are no anomalies in the audit trail. This requirement is the same regardless of where the system resides, on-premise or in the cloud.

## Integrity Controls

How does one know that data initially placed into a database or retrieved in the form of an encrypted file has not been altered?

You might think that the encryption mechanism itself is sufficient but that is not true. Imagine that a hacker has compromised your database and recognizes that the data is encrypted and not useful to him. This doesn't prevent him from causing havoc by copying the values from one record to another. The value copied is altogether a valid, encrypted field; but it has indeed been altered.

One strategy for handling this is to store a long value column with a hash of the record (or file) to validate that the data is the same as was originally written or saved. An insert, update or delete stored procedure should generate a hash of each record that contains PHI, then save that hash in the production database itself in a hash column. The combination of the data in a row (including the identifying fields) and the secret hash key generates a unique long value. The hash key itself can be retrieved either from a key store or from an X.509 certificate uploaded directly into the Management Portal.

The Data Access Layer (DAL) can validate each row against the intra-row hash key before returning the data to be displayed.

It is not possible to guess at what the generated hash column value should be considering that a secret key (the X.509 certificate) is used to generate it.

A scheduled job can run from time to time comparing the hashes from production against expected hashes. If a discrepancy is found, then you can investigate where some person or process outside of your own DAL may have updated data<sup>iii</sup>.

## Person or Entity Authentication

The identity requirement applies to application servers just as it applies to users. This spec can be addressed by making certain that application servers do not communicate amongst themselves except through SSL connections while using a certificate to certify identity.<sup>10</sup> This prevents an identity ID from being embedded in a configuration file, as x.509 certificates can be uploaded and securely stored directly in Azure through the ACS Management Portal<sup>11</sup> and then referenced in code or a configuration file through its thumbprint of the public key.

## Transmission Security

### Integrity Controls

Use secure protocols – HTTPS, SFTP(SH) and FTPS(TLS/SSL). These protocols are all built on SSL.

When connecting to SQL Azure or an on-premise database, specify `Encrypt=True` and `TrustServerCertificate=False` in the database connection string. In fact, SQL Azure does not accept an unsecured connection mechanism. It will attempt to convert it to a secured connection.

Do not depend exclusively on the encryption of files themselves for security. Even encrypted files should be sent over secure transmission mechanisms.

### Encryption

When transmitting Web data or Web Service requests and responses, use SSL. SSL encrypts the data before transmitting it. When transmitting files, use PGP Encryption with the standard, default security configuration. This is AES 256 bit encryption and provides encryption and protection against alteration (integrity).

The HHS Guidance points to National Institute of Standards and Technology documents that proscribe specific lists of encryption technologies that can be used. These and ONLY these may be used.

Hashes are not sufficient to satisfy the HIPAA encryption requirement as brute-force methods can crack them quite easily.

---

<sup>10</sup> <http://msdn.microsoft.com/en-us/library/gg185924.aspx> - Adding an X.509-based Identity Certificate to ACL

<sup>11</sup> <http://msdn.microsoft.com/en-us/library/gg429783.aspx> - Working with the ACL Management Portal

---

## Additional HHS Guidance

### Disaster Recovery

**HIPAA Standard:** *Establish (and implement as needed) policies and procedures for responding to an emergency or other occurrence (for example, fire, vandalism, system failure, and natural disaster) that damages systems that contain electronic protected health information.*

NIST Guidance to standards for providing disaster recovery for a system containing PHI indicates that you must make offsite backups and provide a failover mechanism. This might be reverting to a manual system, but most likely indicates that a sufficiently large organization must perform backups for all data and have a defined process for system recovery in place.

In a cloud system that stores PHI in a SQL database like SQL Azure, the data is backed up to three locations *within the same datacenter*. There are no backups performed *per se*, and nothing is sent offsite. While the likelihood of an entire datacenter going dark is highly unlikely, it is not impossible. So what are the disaster recovery capabilities of SQL Azure?

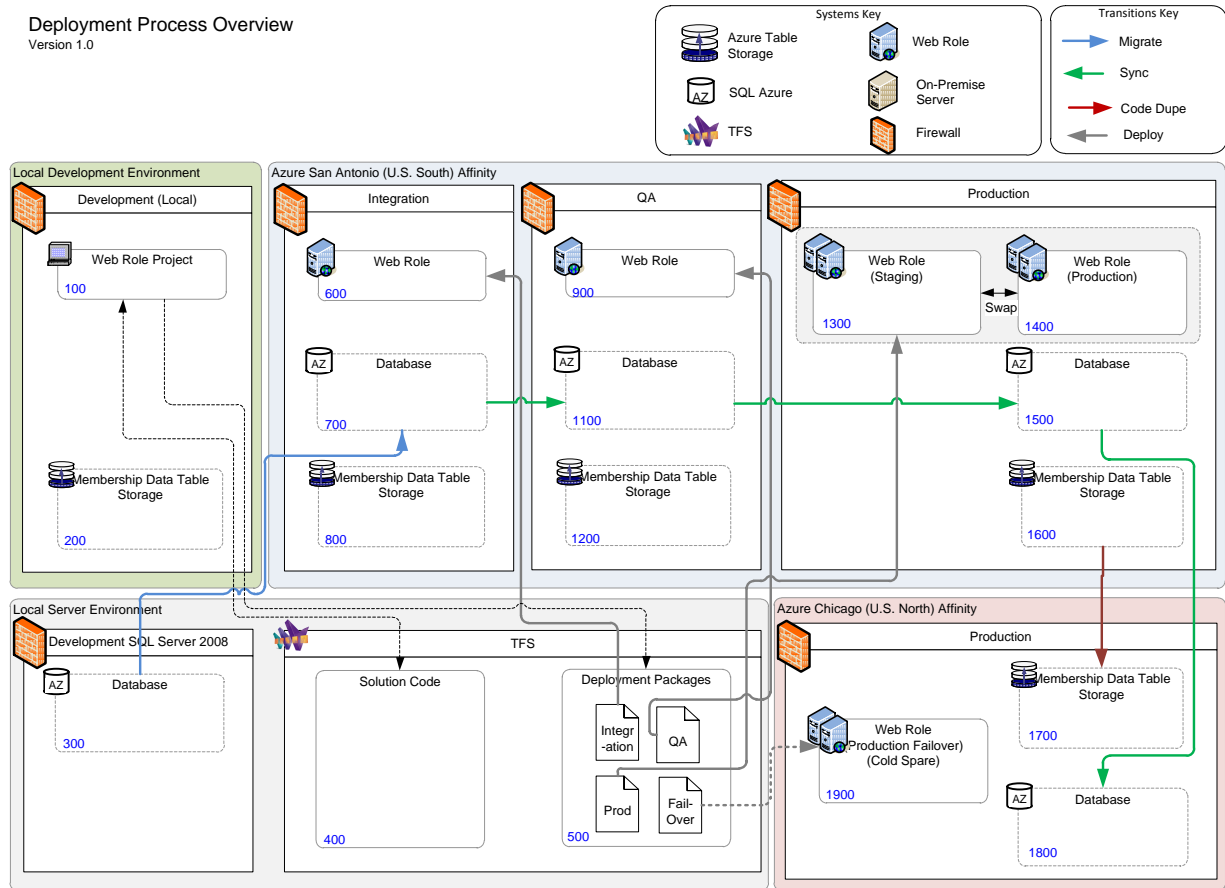
You have three options:

- You can perform a bcp export of the data<sup>12</sup>
- You can run a live, second instance of SQL Azure and Sync the data between the production and failover instances.
- Or, you can back up the SQL Azure instance to a blob in a storage account with a physical affinity in a different datacenter.

The following diagram illustrates a typical Azure deployment process that includes a live failover feature.

---

<sup>12</sup> <http://weblogs.thinktecture.com/cweyer/2011/01/automating-backup-of-a-sql-azure-database-to-azure-blob-storage-with-the-help-of-powershell-and-task-scheduler.html> - Backup Azure Database to bcp's in blob storage.



**Figure 3 - Typical Code and Data Migration with Production SQL Azure Failover**

In this deployment strategy, code is developed<sup>100</sup> against a local on-premise SQL Server 2008 database<sup>300</sup>. This architecture uses Azure Table Storage<sup>200,800,1200,1600,1700</sup> for the Membership provider. Multiple developers work locally against a local version of the database. The project is compiled into 4 deployment packages, each with the connection to the appropriate database. Unlike in a traditional ASP.Net deployment code migration, code itself is not migrated from one platform to another. Rather, at compile time 4 separate deployment packages are created from the same code base, each with a differential Web.Config that points to the database and Membership store for that target platform. All of these deployment packages are placed in the Source Repository, in this case TFS, ready for future deployments as the release moves towards production.

**Recommendation:**  
 When table structure changes, you should perform a Migration.  
 When the structure is stable and you are migrating data, you should synchronize.

**Database Migration vs. Synchronization vs. Backup**

An Azure database can be backed up by performing a BCP export of each table's data. This grabs the data but not the structure of a database. An excellent tool for database migration is an open source project called the Sql Azure Migration Wizard. This tool can be used interactively or from the command line. It analyses the source

database for compatibility with SQL Azure, generates the DDL and exports the data using BCP. This 'Package' is then ready to use to perform a database restore if necessary.<sup>13</sup>

The Migration Wizard (MW) can accommodate small changes in the database model, such as additional tables or columns. Major changes to the database design require the target database tables to be dropped and recreated. For this reason the Migration Wizard is appropriate for the initial deployment of the development database to the cloud, in this case, the migration from 300 to 700.

From there forward in the deployment plan, SQL Azure Sync is more appropriate. Sync is designed to replicate tables and data from SQL to SQL. The database may be synchronized from Integration to QA with the test and supporting decode table data intact. The synchronization from QA to Production would initially be the structure only and additionally only some supporting decode tables.

At this point the production database begins filling with real data. The question is, how do you back it up off-site? This can be accomplished directly with BCP calls, or with the command-line version of the Migration Wizard. This option would require an administrative process to be in place with training and policies that could be followed to restore the database to an alternate Azure facility. This process is HIPAA compliant because this requirement is designated as 'Addressable'.

SQL Azure can now also be backed up to and restored from Azure blob storage. The target Azure storage container can be designated to be housed in a different datacenter (geographic affinity). This does give you a scriptable, offsite backup and restore mechanism.

Your risk audit may well determine that your business can withstand the downtime associated with a manual restore disaster recovery model. But you may instead want a live copy of the database available for near-instant recoverability. This is set up using SQL Azure Sync. Sync is set up to operate on a timer to synchronize the production database to the hot failover database.

A separate Azure deployment package called Failover is created for the Web UI application code, compiled into a package ready to deploy that already contains the connection parameters to the Failover database.

We actually recommend a combination of both strategies. The Sync feature protects you from a datacenter disaster. Storage backup, BCP or the Migration Wizard protects you from yourself.

Following one or the other of these disaster recovery options will demonstrate that you are in compliance with the disaster recovery requirements of HIPAA.

---

## Additional Guidance

### Implement a Key Store

NIST Guidance dictates that encryption algorithms that implement public/private key pairs should be used over hash mechanisms. But it does not specify how one would protect the key pairs themselves. Likewise database connection strings often contain extremely sensitive information. sFTP sites that may be the source or target for interacting with partners need secured username and password pairs. Password salt and hash keys need to be stored somewhere.

**Recommendation:**

Implement a Key Store in the Cloud.

The traditional method of distributing these keys is for the developer to embed them in the configuration files of the application itself. In a traditional datacenter this would allow them to be changed without redeploying the app but this is not a benefit in a cloud application as the role itself could be taken down at any time, destroying whatever configuration changes had been made.

Liken this to an auto dealer where the keys to the inventory of vehicles may be:

1. On the dash of each unlocked car.
2. Distributed and held by the last salesperson who needed them.
3. In a cabinet in the sales manager's office, who holds the key to the cabinet.

Your goal is to get as close to number '3' as possible. A key store is the solution.

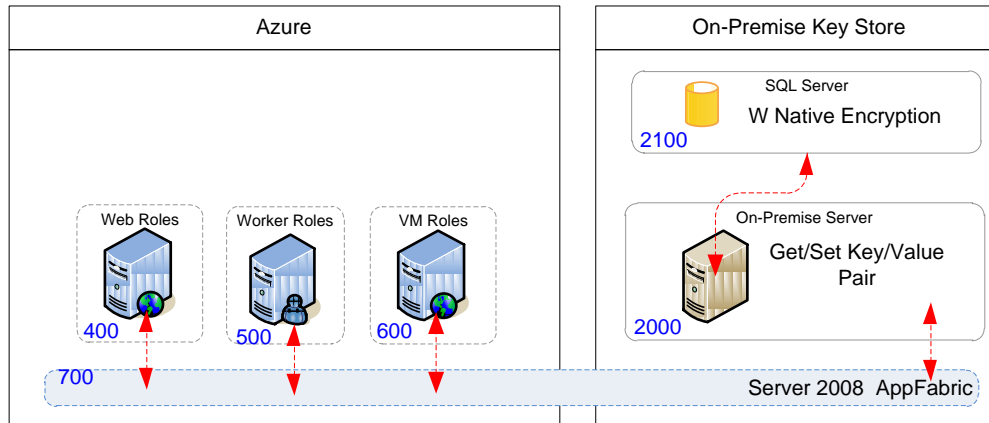
You may want to implement your key store as on-premise or in-the-cloud.

### On-Premise Key Store

An on-premise key store takes advantage of the native SQL Server 2008 R2 ability to encrypt the database.

Worker and Web Roles that need to make connections to various Web Services within a solution get most of their connection information from web.config and app.config entries. But the sensitive information such as certificates, username and password pairs and database connection strings are stored and managed remotely.

This technique can be combined with application-level encryption for added security.



**Figure 4 - On-Premise Key Store**

Advantage:

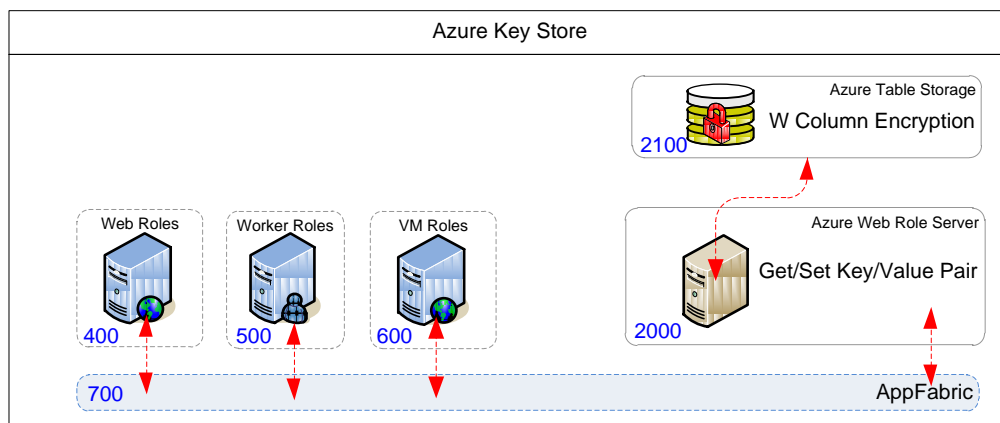
- Centralized Control of Keys
- Backups and Restores Piggyback on Existing Processes

Disadvantages:

- Tying Overall Performance to your Own Datacenter Network’s Reliability
- Network Latency
- Subject to Man-in-the-Middle Intrusion

**In-the-Cloud Key Store**

It is possible to configure a key store in the cloud itself that is as secure as one implemented on-premise.



**Figure 5 - Azure Table Storage Key Store**

In this configuration the various Roles in your solution connect (directly or through an AppFabric connection) to a Web Service that implements a simple retrieval of key/value pairs. The Web Service connection is implemented through an HTTPS connection, with self-signed X.509 certificates at all connection points and that implements presentation of identity through an ACS namespace.

Advantage:

- Speed. All Operations Take Place within the Cloud

Disadvantages:

- The Key Store must be Backed Up Programmatically

## Encrypt Configuration Files

If you do not implement a key store then you should at least encrypt the sensitive information in your web.config and app.config files. This again involves the generation of a self-signed X.509 certificate that is uploaded directly into the Azure Management Portal and used as the key for encrypting and decrypting sensitive information in the configuration files.

A good, multi-part tutorial on this can be found on the Microsoft SQL Azure Team Blog<sup>14</sup>.

## Encrypt Output Files to Hidden Azure Blobs, then Expose them via an HTTPHandler

If your system generates outputs of any kind that are then delivered to the end user through a Web UI interface, then the files themselves should be encrypted before being stored. This is relatively easy and can be done using a streaming interface, where the output of one process is fed into the input of the encryption process and ultimately directly into Azure Storage without ever landing in an unencrypted form.

You can use the CryptoAPI or a commercial product<sup>15</sup> to perform the encryption step.

When you configure the Azure Storage account you should specify that the container not be publicly accessible. This makes it available only to a process that knows the Storage security key and then is doubly protected by its encryption.

The blob is then made available through a special HTTPHandler implemented in the Web UI. A handler is a piece of code designed to handle ASP.Net calls with a particular signature, such as all calls ending in .AzureBlob. A parameter list on the URL can specify the unique identifier of the blob in storage.

When a request for an asset comes in (perhaps from a click on a generated link on the UI), the request is authenticated and secured through the same mechanism and SSL certificates as any other page. The handler parses the blob guid from the URL, retrieves the blob from storage, decrypts it, and streams it back to the user through the response object.

## Encrypt Working Files

It is often necessary to store files on a working drive temporarily when processing them. While most of these operations can be re-written to operate on streams, some may be provided by a third party wherein this is not an option.

The problem you must avoid is in storing the file, even transiently, in an unencrypted state. As soon as you decrypt it and store it, it becomes 'Data-at-Rest'.

You should use a virtual drive in this scenario. The reference architecture for this scenario includes a commercial product called Solid File System from Eldos<sup>16</sup>. This product is a software-based virtual

---

<sup>14</sup> <http://blogs.msdn.com/b/sqlazure/archive/2010/09/07/10058942.aspx> - Encrypting Configuration Files in Azure

<sup>15</sup> <http://www.eldos.com> – Encryption Software Vendor

<sup>16</sup> <http://www.eldos.com/solfs/index.php> – Eldos Solid File System

drive implemented in code. The storage mechanism itself may be memory, scratch storage or Azure Storage implemented as addressable storage<sup>17</sup>.

This storage can be implemented in a format that encrypts the storage (in-memory or otherwise) using HIPAA-compliant AES-256.

It can also be configured to be accessible from only specific Processes, such as the one currently running, or one that you launch from code and for which you obtain the Process ID.

Let's say that you have a need to run an image enhancement process written in Java by the scientific community. While you could port the application to J#, the most direct approach may be to implement a virtual drive in memory, retrieve and unencrypt the source image from Azure Storage and store it on the virtual drive. Then prepare to launch the Java application from code, obtaining the Process ID before actually executing the command. Tell the virtual drive software to trust this process ID, and execute the Java application.

At this point the Java application sees the storage as an attached drive, optionally mapped to a drive letter.

The entire thing is happening in memory, encrypted, and disappears when the process completes.

Using this technique assures that Data-at-rest is never left sitting in storage or on a drive in an unencrypted state.

## Satisfying the Business Associates Agreement Requirement

The Health Information Technology for Economic and Clinical Health (HITECH) Act, enacted as part of the American Recovery and Reinvestment Act of 2009, expands on the requirements set forth in HIPAA to be more explicit about the responsibilities of so-called Business Associates performing services on behalf of Covered Entities. While HIPAA recommended that BA's make the same commitments to protect data as the Covered Entities, HITECH makes these agreements mandatory.<sup>18</sup>

A Business Associate is an entity that receives unprotected PHI on behalf of a Covered Entity. The key phrase here is unprotected. Consider that an Internet Access Provider is not required to sign an agreement with the CE because only encrypted data is transmitted over their network (through the SSL layer). But a company that holds a database of unencrypted PHI clearly must sign a Business Associates Agreement (BAA).

However, it may be possible to construct a system that does not require a BAA if the following conditions are met:

- The PHI is encrypted outside of the cloud.
- The encryption keys are stored in an on-premise key store.
- Unencrypted PHI is never stored or handled in the cloud.

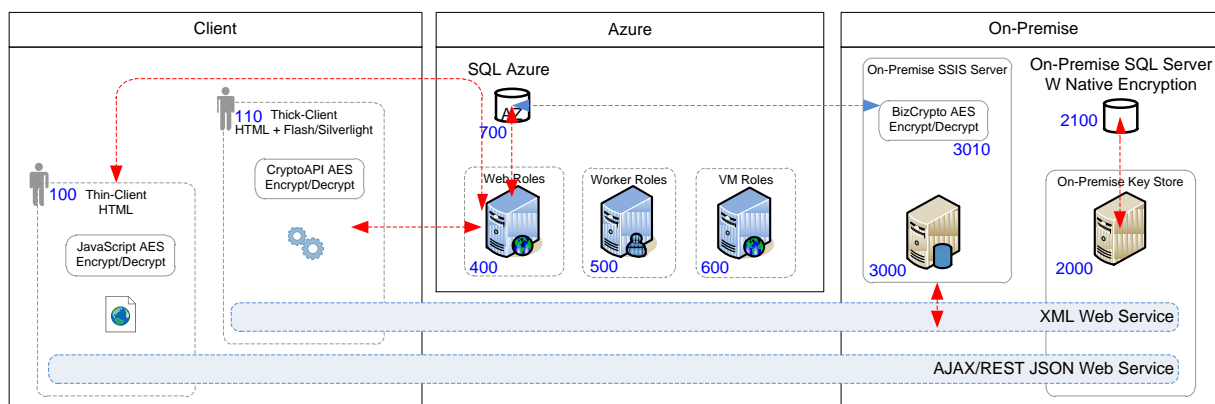
It is possible to architect a system in which PHI is encrypted before being loaded into the cloud, then decrypted outside of the cloud just prior to being displayed. If PHI is changed, it can be encrypted in the client app or intra-browser before being submitted to the cloud for storage.

---

<sup>17</sup> [http://www.eldos.com/solfs/articles/6848.php?sphrase\\_id=265375](http://www.eldos.com/solfs/articles/6848.php?sphrase_id=265375)

<sup>18</sup> There is some debate as to whether existing agreements must be re-signed, or if the HITECH law itself covers the BA's automatically. At any rate the BA's are now subject to the same requirements and reporting triggers regardless of if they are codified in an agreement.

The following architecture achieves this design goal:



**Figure 6 - Client Cryptography Architecture**

The example demonstrates a Web architecture based on Azure with Web<sup>400</sup> and Worker<sup>500</sup> Roles that stores its data in SQL Azure tables<sup>700</sup>.

### Encrypting Initial ETL Data Load

It assumes that the system replaces an existing system and so there is a requirement to perform an initial ETL data load that contains PHI. In this architecture this is performed via an SSIS package with an add-on product that performs AES encryption. The load operation first connects to the on-premise key store to retrieve the encryption key pair, uses the key pair to set the encryption key and then performs a traditional ETL against a data source. The encryption plug-in<sup>3010</sup> encrypts only the PHI columns and submits them for storage in SQL Azure. Note that because this takes place on-premise, unencrypted PHI is never present in any form upon the initial load of data into the system.

Clients interact with the system (perhaps it is an Electronic Health Record (EHR)) through either a thin-client interface or a Rich Internet Application (Flash or Silverlight). Two different strategies exist for crypto services at the client level for these scenarios.

### Thin-Client Cryptography

In the reference architecture data is stored and returned to the page in its encrypted form. Upon receipt of the data (likely in JSON format), a JavaScript AES algorithm<sup>19</sup> can be used to decrypt the PHI fields before being displayed.

The algorithm will need to retrieve the AES key from the key store. For thin-client implementations this can be done upon page load by making an AJAX call to the on-premise key store through an AJAX proxy or other cross-domain AJAX handling technique like JSONP.<sup>20</sup> Using these techniques data can be retrieved into JSON structures, the PHI fields decrypted and then displayed. Updates to PHI fields go through the reverse process to encrypt them before submitting to the server for storage.

<sup>19</sup> <http://point-at-infinity.org/jaaes/>

<sup>20</sup> [http://abhinavsingh.com/blog/2009/11/making-cross-sub-domain-ajax-xhr-requests-using-mod\\_proxy-and-iframe/](http://abhinavsingh.com/blog/2009/11/making-cross-sub-domain-ajax-xhr-requests-using-mod_proxy-and-iframe/)

## RIA-Client Cryptography

Many green-field development projects today utilize Rich Internet Application client plug-ins like Flash and Silverlight. These technologies are more direct in their abilities to make cross-platform Web Service calls. The techniques are the same as for thin-client but for Flash and Silverlight a cross-domain file must be placed on the Web server that explicitly authorized the client plug-in to make a call to a domain different from the one that returned the page itself.

For Silverlight consider using the AESManaged library<sup>21</sup>.

For Flash consider using the open source as3crypto library<sup>22</sup>.

## Searching on Encrypted Data

One consequence of storing encrypted data is that new strategies must be developed for searching and reporting against the encrypted fields. A member's name or membership number is PHI, but it is also likely that an accountant or clerk would need to locate a member by these criteria.

A search mechanism can be designed by encrypting the full search term (such as member ID) before submitting. Then your SQL query can compare the encrypted search term against the encrypted values stored in the member ID field, returning the record for the desired member.

This technique may provide all of the search capabilities you need, but you will indeed lose the ability to perform partial term or wildcard searches.<sup>23</sup> This is an engineering tradeoff that must be considered when deciding whether to place your application in the cloud.

## Reporting on Encrypted Data

Another engineering compromise you must keep in mind if you go the route of end-to-end client-level encryption is to consider the effect that this technique will have on your reporting requirements.

Many of our clients design their reports in Reporting Services which is now available in SQL Azure.

Some reports will work against encrypted data while some will not. Reports that contain PHI, like patient schedules or reimbursement reports, cannot be generated against encrypted data. Reports that are aggregated by non-PHI data will naturally not include the PHI itself but the grouping element cannot be PHI. For example, zip codes and state are protected elements, and so you could not produce a report (on the server) that summarizes claims or patient counts by state or zip.

## *Thin-client Client Reporting*

As in many client/server concepts many things that can be done on the server can also be done on the client. In this case you could consider returning data sets to JavaScript in JSON format, unencrypt the PHI using the JSAES library, then construct the report directly in HTML using a JavaScript Grid like that provided in the Sencha toolkit.<sup>24</sup> This advanced grid includes searching, grouping, summary rows, calculated fields, etc.

---

<sup>21</sup> AESManaged is available within the System.Security.Cryptography namespace of Dot.Net.

<sup>22</sup> <http://code.google.com/p/as3crypto/>

<sup>23</sup> Like 'Smi\*' or 'S?ith'.

<sup>24</sup> <http://www.sencha.com/products/extjs/examples/#sample-2>

One drawback of a JavaScript report constructed from a grid like Sencha would be that the resulting grid may not print exactly the same on all client machines due to resolution differences. If exact report layout is desired you may want to consider a hybrid approach that uses JavaScript and AJAX to retrieve and unencrypt the data but that uses a thick client reporting tool in Flash to construct and display the report itself, like the Hexatech Viewwidget product.<sup>25</sup>

### ***Thick-client RIA Client Reporting***

Reporting is easier in RIA tools like Silverlight but you must be careful that the reporting tool you decide on is not just a report rendering component. These tools must create the report on the server where PHI is encrypted. Tools like Report Sharp-Shooter<sup>26</sup>, ComponentOne<sup>27</sup> and Telerik<sup>28</sup> fall into this category.

Tools such as DevExpress Reporting<sup>29</sup> and SyncFusion Reporting<sup>30</sup> tools separate layout from the datasource, allowing the datasource for a report to come from a Web Service or object model where there is an opportunity to intercept the results to allow for decryption.

---

<sup>25</sup> <http://www.hexatech.com>

<sup>26</sup> <http://www.perpetuumsoft.com/Report-Sharp-Shooter-for-Silverlight.aspx>

<sup>27</sup> <http://www.componentone.com/SuperProducts/ReportViewerSilverlight>

<sup>28</sup> <http://www.telerik.com/products/reporting.aspx>

<sup>29</sup> <http://devexpress.com/products/net/reporting/silverlight.xml>

<sup>30</sup> <http://www.syncfusion.com/products/reporting-edition/report-viewer>

---

## HIPAA Technical Safeguards

(Actual rule from HHS)<sup>31</sup>

### § 164.312 Technical safeguards.

A covered entity must, in accordance with § 164.306:

(a)(1) *Standard: Access control.* Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in § 164.308(a)(4).

(2) Implementation specifications:

(i) *Unique user identification* (Required). Assign a unique name and/or number for identifying and tracking user identity.

(ii) *Emergency access procedure* (Required). Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency.

(iii) *Automatic logoff* (Addressable). Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity.

(iv) *Encryption and decryption* (Addressable). Implement a mechanism to encrypt and decrypt electronic protected health information.

(b) *Standard: Audit controls.* Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

(c)(1) *Standard: Integrity.* Implement policies and procedures to protect electronic protected health information from improper alteration or destruction.

(2) *Implementation specification: Mechanism to authenticate electronic protected health information* (Addressable). Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.

(d) *Standard: Person or entity authentication.*

Implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed.

(e)(1) *Standard: Transmission security.* Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.

(2) Implementation specifications:

(i) *Integrity controls* (Addressable). Implement security measures to ensure that electronically transmitted electronic protected health information is not improperly modified without detection until disposed of.

(ii) *Encryption* (Addressable). Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate.

---

<sup>31</sup> <http://www.hhs.gov/ocr/privacy/hipaa/administrative/combined/index.html>

Part 164: [http://www.access.gpo.gov/nara/cfr/waisidx\\_07/45cfr164\\_07.html](http://www.access.gpo.gov/nara/cfr/waisidx_07/45cfr164_07.html)

Technical Safeguards: [http://edocket.access.gpo.gov/cfr\\_2007/octqtr/pdf/45cfr164.310.pdf](http://edocket.access.gpo.gov/cfr_2007/octqtr/pdf/45cfr164.310.pdf)

---

## Summary

While placing PHI in the cloud is a challenge, the technological hurdles are not insurmountable. Reasonable precautions as described in this paper will keep you in line with the HIPAA law. And if your system is ever compromised you may not even be required to report the incident.

Many of the HIPAA requirements involve administrative, process and policy steps that you must take regardless of where your application sits.

Data-in-motion and data-at-rest must be encrypted using specific technologies, and a hash is not sufficient protection for data itself.

---

## About the Author

Mick Seals is a Senior Manager Consultant and the Azure Solution Area Lead for the Southwest region of Sogeti, USA, LLC. Mick is available for consulting, speaking engagements and writing opportunities. He has been published in the peer-reviewed journal JHIM, the Journal of Healthcare Information Management.<sup>iv</sup>

He can be reached directly at:

[michael.seals@us.sogeti.com](mailto:michael.seals@us.sogeti.com)

Or through the Sogeti Dallas office:

Mick Seals c/o  
Sogeti USA LLC  
222 Las Colina Blvd W  
Suite 1550  
Irving, TX 75039

## About Sogeti USA, LLC

Sogeti is a workforce of more than 20,000 people operating in 15 countries; Belgium, Denmark, Finland, France, Germany, India, Ireland, Luxemburg, Netherlands, Norway, Spain, Sweden, Switzerland, UK and the US.



Sogeti was awarded global Microsoft Enterprise Partner of the Year for 2010.



**2010 PARTNER OF THE YEAR**  
Enterprise  
Winner

---

## Related Links

See the following resources for further information:

- Guidance regarding Data Use Agreements for research and public health:  
<http://www.hsrmetho ds.org/PrivacyInResearch/Privacy%20Tools/Guidance%20on%20HIPAA%20Data%20Use%20Agreements.aspx>
- The web site you want to avoid: <http://transparency.cit.nih.gov/breach/index.cfm>

For the latest in Sogeti technology initiatives, please contact Sogeti USA at <http://www.us.sogeti.com>

---

<sup>i</sup> 'Secured' as defined by the HIPAA regulation guidance issued by HHS.

<sup>ii</sup>

<http://www.hsrmetho ds.org/PrivacyInResearch/Privacy%20Tools/Guidance%20on%20HIPAA%20Data%20Use%20Agreements.aspx> This link contains a specific list of data elements that are considered to be 'Identifying'. The remaining health information can be used only for medical research, public health, or health care operations.

<sup>iii</sup> While the encryption standards are proscribed out of a list of specific algorithms that can be used (like AES), this is not the case with integrity checks. MD5 or SHA1 would be sufficient for this purpose.

<sup>iv</sup> <http://www.scribd.com/doc/50191947/The-Use-of-XML-in-Healthcare-him14208> - Journal of Healthcare Information Management