Protecting highly confidential sales data with Azure SQL Database
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protecting highly confidential sales data with Azure SQL Database</td>
<td>1</td>
</tr>
<tr>
<td>The evolution of DDA architecture</td>
<td>1</td>
</tr>
<tr>
<td>Network security</td>
<td>2</td>
</tr>
<tr>
<td>Network security challenges</td>
<td>4</td>
</tr>
<tr>
<td>Authentication</td>
<td>4</td>
</tr>
<tr>
<td>Authorization</td>
<td>5</td>
</tr>
<tr>
<td>Encryption</td>
<td>6</td>
</tr>
<tr>
<td>Security monitoring</td>
<td>7</td>
</tr>
<tr>
<td>Securing apps in the future</td>
<td>8</td>
</tr>
<tr>
<td>Putting it all together</td>
<td>8</td>
</tr>
<tr>
<td>Appendix</td>
<td>9</td>
</tr>
<tr>
<td>For more information</td>
<td>10</td>
</tr>
</tbody>
</table>
Protecting highly confidential sales data with Azure SQL Database

At Microsoft, Core Services Engineering (CSE, formerly Microsoft IT) is a close partner with the product teams. As a custodian of some of the company’s most sensitive data, we are held to high security standards. We use the built-in capabilities of Microsoft Azure SQL Database to meet stringent security controls for storing sensitive data in the cloud. Azure SQL Database applies our security requirements, helps strengthen our database security in the cloud, and allows us to use the Platform as a Service (PaaS) cloud model.

Azure SQL Database offers a broad suite of enterprise-grade security tools to help with authentication and authorization, database encryption, data access policies, and proactive database monitoring. We use Azure SQL Database to protect an important internal sales tool—Microsoft Enterprise Deal Discount Approval (DDA).

The DDA tool is a classic multi-tier application. DDA controls the approval process for licensing deals and discounts at our enterprise sales organization in France. The application, running on the Windows 10 Universal Windows Platform, helps simplify the discount approval workflow and offers improved discount control at the subsidiary.

The information processed by DDA, including discounts and other non-pricing concessions, is extremely sensitive. In fact, such data is labeled Highly Confidential, and “represents very sensitive business data which would certainly cause business harm if over-shared,” according to Microsoft data classification standards. We maintain security controls and requirements that any application processing Highly Confidential data needs to adhere to.

The DDA team that manages this tool needed first-class security measures as it moved this sensitive data to Azure SQL Database.

The evolution of DDA architecture

The DDA team started with simple architecture:

- Azure SQL Database to store structured data
- Azure Storage Account (Blob) to store files
- Azure Storage Account (Queue) for messaging
- App Service for the API layer—REST API
- Mobile client
Figure 1 shows the simple architectural structure we started with.

![Diagram showing simple architecture with Storage blob, Application data, REST API, Mobile App, and Storage queue]

*Figure 1. DDA began with simple architecture*

Over time, this architecture evolved to meet more stringent controls around network security, authentication, authorization, encryption, and security monitoring.

**Network security**

The threat model of the original architecture exposed risks. One risk is that both Azure SQL Database and Azure Blob Storage are accessible via public IP addresses. Both data stores contain sensitive information, so it is important to restrict access to them.

To achieve that, the DDA REST API App Service was deployed within an App Service Environment (ASE) inside a virtual network. The outbound address used by REST API to communicate with Azure Storage and Azure SQL Database is the public virtual IP (VIP) of the ASE. The VIP is a static IP address that can be found in the portal user interface for the ASE in the Properties settings. (See [Network Architecture Overview of App Service Environments](#) for details.)

Once the outbound address of the REST API is fixed, the Azure Storage and Azure SQL Database endpoints can be secured as follows:

- **Azure SQL Database firewall rules** can be set up so that incoming connections to the DDA database are allowed only from the REST API's fixed IP address. Without a fixed IP address, the firewall rules would need to allow connections from all Azure services, which would increase risk.

- The fixed IP address of the REST API service is set inside a Security Access Signature (SAS) key that the REST API uses to connect to Azure Storage. The SAS key cannot be used from any other IP addresses. ([Using Shared Access Signatures (SAS)](#) has more information about SAS.)
At this point, our basic DDA topology was enhanced by an ASE with a fixed IP address and the Azure SQL Database firewall, shown in Figure 2.

**Figure 2. Enhanced DDA architecture**

Since DDA users need to connect to the DDA REST API from the internet, this topology leaves the REST API App Service as a public endpoint. REST API requires very high-level privileges, including the ability to decrypt sensitive data stored in the database. If the REST API is compromised, all data could be accessed in cleartext. To remediate this risk, the network topology was further enhanced to use the Gatekeeper pattern with another ASE to isolate inbound traffic to the REST API, as shown in Figure 3.

**Figure 3. Enhanced DDA architecture with Gatekeeper**
Network security challenges
While this architecture meets the security requirements, it affects:

- **Performance.** Adding an extra Gatekeeper layer to isolate the network increases latency for client requests. Furthermore, multiple authentication roundtrips increase the response time when new authentication is required.
- **Manageability.** Both App Services may have to scale to accommodate a large volume of requests.
- **Supportability.** Again, the extra Gatekeeper layer increases complexity. Instrumentation must be carefully planned to enable proper root cause diagnosis.
- **Cost.** App Service Environments are a key element of securing the application; however, they can be costly for small applications. Consider using the ASE for multiple applications.

It is important to weigh the benefits against the disadvantages to balance tradeoffs that are consistent with your policies.

Authentication
DDA uses Azure Active Directory (Azure AD) to authenticate users, operators, and service-to-service authentication. The authentication flow, which is based on [Authentication Scenarios for Azure AD](#), is shown in Figure 4.

![Authentication diagram](image_url)

**Figure 4. The authentication flow**

For more information about the authentication flow, see:

- Application to Gatekeeper as described in [Native Application to Web API](#).
- Gatekeeper to REST API as described in [Web Application to Web API](#) (using Delegated User Identity with OpenID Connect).
• Secrets to connect from REST API to Azure SQL Database and Azure Storage Account are stored in Azure Key Vault as described in Use Azure Key Vault to protect application secrets.
• REST API to Azure Key Vault as described in Use Azure Key Vault from a Web Application (Authenticate with a Certificate instead of a Client Secret).
• REST API to Azure Storage as described in Using shared access signatures (SAS).
• REST API to Azure SQL Database as described in Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse.

The REST API employs a trusted subsystem model, which means it connects to the database using a single database principal. To achieve non-repudiation, we obtain the user’s identity from the claim information in the on-behalf token and pass it in all the CRUD database operations via a stored procedure parameter.

An alternative approach is for the REST API service to impersonate users against the database. To do that, the REST API could open a token for Azure SQL Database on behalf of each application user (like what Gatekeeper does). One advantage of this approach is that the database audit log captures actions of individual application users directly, without relying on data passed as a Stored Procedure parameter. However, it also requires granting users access to the database and to keys in Azure Key Vault (see the Encryption section of this paper). We felt that this process poses a threat that we wanted to avoid.

Authorization

The REST API can connect to the Azure SQL database, because it contains a database user that is associated with the Azure AD principal representing the REST API. That database user has the CONNECT database permission, as well as other permissions required for the REST API to support its use.

The database user for the Azure AD principal has been created using this Transact-SQL statement:

```
CREATE USER [<username@microsoft.com>] FROM EXTERNAL PROVIDER;
```

The permissions configuration in the database follows the least-privilege principle. All database users, including the user representing the REST API, are granted the minimum permissions needed for their jobs.

```
CREATE ROLE [DDA_ProcedureExecution] AUTHORIZATION [dbo];
GO
GRANT EXECUTE ON DDA_UpdateDealDetails TO [DDA_ProcedureExecution];
GO
GRANT EXECUTE ON DDA_UpdatePricingDetails TO [DDA_ProcedureExecution];
GO
GRANT VIEW ANY COLUMN ENCRYPTION KEY DEFINITION TO [DDA_ProcedureExecution];
GO
GRANT VIEW ANY COLUMN MASTER KEY DEFINITION TO [DDA_ProcedureExecution];
GO
ALTER ROLE [DDA_ProcedureExecution] ADD MEMBER [<username@microsoft.com>];
```

The first two GRANT statements enable the DDA_ProcedureExecution role to execute stored procedures that are required for supported REST API uses. The role has no direct permissions on the underlying views and tables, which means the middle tier cannot execute arbitrary queries against those views and tables.

Two more GRANT statements give the role access to Always Encrypted key metadata, which is required to query encrypted data. (Always Encrypted is described in the next section.)
The database also contains a database user that maps to an Azure AD group, representing the DDA Operations team. (we are using the `<DDAOperationsGroupName>` placeholder for the group name in the below script).

```
CREATE USER [<DDAOperationsGroupName>] From External Provider;
```

This database user is a highly privileged principal that is permitted to directly read and write data, execute stored procedures, and alter Always Encrypted key metadata.

```
ALTER ROLE [DDA_ProcedureExecution] ADD MEMBER [<DDAOperationsGroupName>];
ALTER ROLE [db_datawriter] ADD MEMBER [<DDAOperationsGroupName>];
ALTER ROLE [db_datareader] ADD MEMBER [<DDAOperationsGroupName>];
GRANT ALTER ANY COLUMN MASTER KEY TO [<DDAOperationsGroupName>];
GO;
GRANT ALTER ANY COLUMN ENCRYPTION KEY TO [<DDAOperationsGroupName>];
GO;
```

Note that the `<DDAOperationsGroupName>` group in Azure AD has no permanent members. Thus, no operator has standing access to the database. To troubleshoot a production issue in the database, an operator must request temporary access via an internal Microsoft access management tool. Once a manager approves the request, the operator’s Azure AD principal is temporarily added to the `<DDAOperationsGroupName>` Azure AD group. The user principal is automatically removed from the group after a short time.

Encryption

Our standards specify that all data at rest must be protected from offline physical and virtual media attacks. To meet this requirement, DDA uses Azure Storage Service Encryption and SQL Transparent Data Encryption (TDE) for Azure SQL Database. TDE encrypts data at the page level. The pages in an encrypted database are encrypted before they are written to disk and decrypted when read into memory. This protects data stored in data files, transaction logs, and in backup files.

Special requirements apply to Highly Confidential data, such as discount amounts and other sensitive sales deal information, that DDA processes. Such data must be encrypted before going to a data store, such as Azure Storage or Azure SQL Database. To meet this requirement, DDA uses Client-Side Encryption for Microsoft Azure Storage and Always Encrypted for Azure SQL Database.

Typically, implementing a client-side encryption solution for a relational database is hard and expensive. It requires a developer to identify all queries that target columns storing sensitive information and to modify the application code to inject encryption and decryption routines before and after each query. Client-side encryption makes it usually impossible for a database system to support any queries or computations on encrypted data. In addition, developers need to implement a solution for provisioning and managing the encryption keys.

Always Encrypted allows DDA developers to avoid those challenges. Always Encrypted is an industry-first technology in Azure SQL Database that transparently encrypts data on the client-side, before the data is sent to the database. At the same time, Always Encrypted ensures that point lookup queries and equality joins are supported on encrypted columns. This capability enabled DDA developers to implement keyword-based search on sales deals, while protecting sensitive deal information from malware or rogue insiders.

DDA stores column master keys in Azure Key Vault – the most convenient option for applications hosted in Azure. The master keys are regularly rotated. The DDA Operations team has automated key provisioning and key rotation by developing PowerShell scripts, following the examples in the documentation: Always Encrypted (Database Engine).
Security monitoring

Enforcing security policies and monitoring security across an organization can be a challenge. The DDA team actively monitors their environment for security events and analyzes those events to determine if they are incidents. By integrating their monitoring system with an array of Azure services, the DDA team makes it easy to respond to a potential incident. (See SQLDatabase.md for more information.)

The DDA Operations team monitors security enforcement in three ways:

- Apply consistent security policies.
- Actively monitor all resources.
- Create tailored alerts to be notified incidents or policy violations.

To apply a consistent security policy to every new SQL database, the operations team uses JSON config files. This template of feature configurations helps to enforce technical control procedures that we must abide by. The JSON configurations execute the Security Development Lifecycle (SDL) Technical Control Procedures mentioned earlier, like enabling SQL Database Audit, Threat Detection, and Transparent Data Encryption.

For database monitoring, SQL Database’s built-in audit logs allow the DDA team to track database events, helping to maintain regulatory compliance, understand database activity, and find discrepancies that could indicate security issues. Additionally, with SQL Audit Logs now stored in Blob storage, the DDA team can benefit from better performance, higher granularity, and lower storage cost for SQL Database Audit.

More broadly across the architecture, logs from the following Azure services are collected and stored in a dedicated storage account:

- SQL audit logs (integrated with Azure Security Center and OMS)
- Subscription audit logs
- Storage analytics
- Network security group diagnostic logs
- Azure Key Vault Diagnostics
- and App Service Audit Logs

Having all this available data is important for forensics but trying to find the right information in Blobs can be time consuming. To facilitate log analysis, most of this data is centralized in a single Operation Management Suite (OMS) workspace. This way, the team can use a unified Search Engine.

Beyond a centralized workspace for analyzing logs, the team relies extensively on automated alerts to notify them of security events. Using scripts provided by the Microsoft security team, the DDA team was able to create relevant Insights based alerts for their services. The team also relies on the built-in proactive monitoring tool, Azure SQL Threat Detection, to notify them of potential malicious activities. SQL Threat Detection runs algorithms to detect potential vulnerabilities, potential SQL injection attacks, and anomalous database access from an unusual location or principal.

For example, with SQL Threat Detection, the team can quickly learn about a suspicious login through an alert in Azure Security Center. In accordance with the team policy, when a team member picks up the investigation of a security alert, they dismiss the alert in Azure Security Center, so the rest of the team knows it’s being addressed. The Threat Detection alert will show the related audit logs around the suspicious activity in OMS, and the team member can gain more insight by analyzing and querying other events that happened around the same time.

Because OMS is the team’s central monitoring tool, not only are database alerts integrated there, but also other Azure service alerts, such as from Azure Key Vault. The team also set their own queries in OMS to create custom alerts. These alerts are tailored to their security policy, and capture events that are non-standard behaviors typically done by an application, not a human. Examples include someone authenticating using SQL authentication (instead of Azure AD authentication) to a SQL database, or someone trying to read data without using a stored procedure. They also have email alerts if something other than their REST Service layer tries to access keys directly in Key Vault (such as through the Azure Portal).
Besides responding to alerts, members of the DDA Ops team regularly check the subscription health and application health dashboards on OMS. These dashboards show the team how well their Azure resources are adhering to CSE policies, which they had translated into controls within OMS.

The DDA Ops team also wants to track any drift that has occurred in their security policies. OMS helps the team keep track of how well their resources are adhering to the security policies that the team defined for their databases (which were shown in JSON config in the beginning of this section).

Similarly, for broader Azure services, the DDA Ops team can use Azure Insights to create rules for Azure-wide operations that would violate their policies. Azure Insights runs on Azure Activity Log and role-based access control, generating alerts for any suspicious activity that manipulates access or important resources. Some of the actions they track are changing TDE configuration, altering firewall or audit rules, and modifying Key Vault access policies.

### Securing apps in the future

We aim to eventually have all applications in PaaS. By moving to PaaS, we can unlock advantages that only the PaaS model offers, such as out-of-box scale-outs and scale-ups, better cost control, and easier support for mobile applications.

The DDA Operations team is constantly looking to improve their application security. Their security goals go beyond our requirements, as they look for more ways to secure their system. For example, one main security focus is to add Virtual Network support. They will also expand their use of Azure Key Vault and adopt Bring-Your-Own-Key support for SQL Transparent Data Encryption, so they can use their own key from Azure Key Vault for encryption-at-rest in their databases.

DDA is also evaluating the new confidential computing capabilities in Always Encrypted to support even richer queries on encrypted database columns and simplify key management via in-place encryption.

Another focus is to improve the monitoring system and overall security manageability as their application and team scales. They are integrating OMS and email alerts into Microsoft Visual Studio Online (VSO), so that VSO can serve as the central monitoring and incident response destination for their Development Operations team (DevOps). They would also like to integrate email alerts into our ticketing system for faster response and better organization. Today, their development environment also receives a high volume of email alerts, so they are working to refine their alerting rules and remove false positives before deploying to production.

For incident response, they plan to improve just-in-time (JIT) access provisioning for the DevOps team by integrating Azure AD Privileged Identity Management with role-based access control. This eases the management, control, and monitoring of access within the team. The team is also continuing to invest in automation by implementing a bot for JIT access requests and working toward unsupervised key and credential rotation.

### Putting it all together

To build strong network security, we enforce fixed IP addresses and only allow traffic from whitelisted addresses using SQL Firewall rules. Two-factor authentication with Azure AD controls the authentication flow through Azure Key Vault, Azure Storage, and Azure SQL Database. SQL Transparent Data Encryption encrypts our databases at rest, and SQL Always Encrypted encrypts the most sensitive sales information and stores the encryption keys on the client side.

The cloud is unfamiliar territory for companies, especially when they consider storing sensitive data. We were able to use the built-in, multi-layered security functionality in Azure SQL Database to meet our own stringent security requirements. By coupling Azure SQL Database's security with the suite of Azure monitoring tools, other companies can employ disciplined security protocols and operational controls while benefiting from the cost savings and manageability of Azure.
Appendix

The Digital Security and Risk Engineering (DSRE) group at Microsoft is a central organization within CSE that, among other services, develops guidelines for internal adoption of technologies. DSRE automation and guidance helps us manage internal applications in two major areas: Security Development Lifecycle (SDL) and Operational Security Assurance (OSA). It also offers consulting services to help create secure service topology. SDL procedures are typically managed and implemented by development teams; OSA procedures are typically addressed by operations teams.

Tables 1 and 2 contain examples of SDL and OSA procedures that helped the DDA team achieve compliance.

Table 1. Sample DDA SDL procedures

<table>
<thead>
<tr>
<th>Category</th>
<th>Applicable procedures</th>
<th>Azure services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Implement proper authentication method</td>
<td><strong>Azure Active Directory authentication</strong> for Azure SQL Database lets us rely on central identity management and central policies (password rotation frequency, password strength policy, etc.).</td>
</tr>
<tr>
<td>Authorization</td>
<td>Implement access control to check user authorization</td>
<td><strong>Contained Database User</strong> from external provider. <strong>SQL Firewall</strong> in combination with <strong>App Service Environment</strong> isolates access to the Azure SQL Database.</td>
</tr>
<tr>
<td>Cryptography</td>
<td>Create and manage keys and certificates securely</td>
<td><strong>Azure Key Vault</strong> provides storage for secrets such as connection strings, Azure Storage <strong>Shared access signature</strong> tokens and encryption keys.</td>
</tr>
<tr>
<td>Data Handling</td>
<td>Secure sensitive data in storage and code</td>
<td><strong>Always encrypted</strong> and <strong>Transparent data encryption</strong> are available for Azure SQL Database.</td>
</tr>
<tr>
<td></td>
<td>Secure sensitive data in transit</td>
<td><strong>SSL/TLS</strong> is available for Azure SQL Database. Furthermore, encryption is enforced by default.</td>
</tr>
<tr>
<td>Category</td>
<td>Applicable procedures</td>
<td>Azure service</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Auditing and logging</td>
<td>Configure appropriate security event logging</td>
<td>Azure SQL Database Auditing provides the ability to log activities for the right duration in durable storage. Auditing is also available for other services used by DDA. Auditing data can be used for custom alerts with Operations Management Suite integration and for forensic analysis if required. Azure SQL Database Threat Detection enables detection and response to potential threats. Azure Security Center (ASC) provides a record of Azure SQL Database Threat Detection Alert as well as a dashboard to monitor for Security Health.</td>
</tr>
<tr>
<td></td>
<td>Appropriate retention of security event logs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Subscribe to or implement appropriate security monitoring</td>
<td></td>
</tr>
<tr>
<td>Cryptography</td>
<td>Use approved Secret Storage Solution</td>
<td>Azure Key Vault offers hardware security modules to protect keys.</td>
</tr>
<tr>
<td></td>
<td>Use hardware security modules (HSM) to protect private keys</td>
<td></td>
</tr>
<tr>
<td>Desired State Configuration</td>
<td>Implement SQL Server Baseline Configuration</td>
<td>Azure Security Center provides the ability to set a number of configuration policies for some the services used by the DDA application.</td>
</tr>
<tr>
<td></td>
<td>Implement Web Server Security Baseline Configuration</td>
<td></td>
</tr>
<tr>
<td>Data Handling</td>
<td>Support Secret and Key Rotation</td>
<td>Always Encrypted support Column Master Key rotation</td>
</tr>
<tr>
<td>Network Isolation</td>
<td>Endpoints enabling access to sensitive data should not be publicly accessible</td>
<td>App Service and Azure Storage Account are public endpoints. Azure SQL Database provides a firewall but to be efficient, fixed IP addresses are needed. ISRM recommended the use of App Service Environment to isolate endpoints from public traffic. This enables configuration of tighter rules on the SQL Server firewall as well as using SignedIP parameter in Storage Shared Access Signature tokens.</td>
</tr>
</tbody>
</table>