dbsign™

Data Security Suite

DBsign Concepts Manual

Gradkell® Systems Inc.

Version 4.0

Table of Contents

1.0 Introduction

1.1 Document Organization

This document introduces concepts related to using DBsign to integrated digital signatures into relational database applications. It is a good place to start reading to understand DBsign prior to starting your integration.

This document is divided into two main sections:

- **Digital Signature and Relational Database Concepts**: Section 2.0 discusses the general concepts related to digital signatures and how they are used and integrated into database-driven applications. This section explains the issues associated with digital signature integration and describes DBsign's solution.
- **DBsign Concepts**: Section 3.0 explains concepts unique to DBsign's digital signature implementation. These concepts are fundamental to understanding how to use DBsign in your application.

1.2 Contacting Gradkell Systems

Web Site

Gradkell's web address is http://www.gradkell.com/.

Email

- Sales inquiries, send email to sales@gradkell.com.
- For DBsign technical support, email support@gradkell.com.

Postal Service

Corporate Headquarters:

4910 University Square, Suite 2
Huntsville, AL 35816

Telephone

Corporate Headquarters:

866-GRADKELL (866-472-3535)

2.0 Digital Signature and Relational Database Concepts

This section was published by Auerback Publications in the Information Security Handbook*, Volume 3, 4th Edition by Tipton and Krause. This is a collection of articles from industry experts in IT security. The content of this section appeared as Chapter 31,* Digital Signatures in Relational Database Applications *by Mike R. Prevost. It covers the issues peculiar to integration digital signature into relational database environments.*

2.1 Introduction

Now that public key encryption and its associated infrastructure (PKI) have become an accepted foundation for securing the electronic world, we are seeing a wealth of new security products come on the scene. However, it seems that many of these products are solving security problems related to the infrastructure upon which business applications run rather than the applications themselves. For example, virtual private network (VPN) products are beginning to support certificate-based authentication and public key based key exchange. SSL is the standard for privacy and authentication on the web. While these types of technologies are completely necessary, they are all highly specialized and are invisible to the applications they are securing.

The nature of digital signature technology and its use in database driven applications requires a certain amount of application integration. It is this integration step that has been the primary technical stumbling block to the widespread use of digital signatures. PKI programming is still a “black art” known only to the few who have conquered its formidable layers of complexity. PKI integration projects have proven too costly and too risky for many application owners. As a result, organizations seem to be focusing on ways to add security to applications without performing complex integrations. However, as we move from securing our infrastructure to securing our applications, there is a growing genre of data security products that are making it easier to integrate security features such as digital signature into the applications themselves.

This section discusses the issues associated with integrating digital signature functionality into relational database applications. First, we discuss some concepts about digital signature and the role that digital signature plays in an application security strategy. Next, we explain why relational database applications are different than other environments and discuss some of the pitfalls of various integration approaches.

2.2 Digital Signature Concepts

In relational database applications, digital signatures are typically used to ensure data integrity and/or non-repudiation (i.e., proof of origin). Since digital signatures are semantically similar to paper signatures, they are used to streamline business processes by reducing or entirely eliminating the need to print, sign, transfer and store paper documents. The legal framework for holding signers accountable for documents they digitally sign is beginning to take shape. Then we outline DBsign's "application generic" solution to digitally signing data stored in relational databases that is very easy to integrate into applications.

2.2.1 The Anatomy of a Transaction

When discussing application security, the term "transaction" is often used. This is a very vague term that brings to mind financial or business transactions. Sometimes the term "document" is used. For our immediate purposes, a transaction (or document) is any exchange between the user and the application that results in a change to data that is stored by the application. In database applications, the transaction data is stored in a relational database.

The following diagram breaks a transaction into four steps. Each step has unique security requirements. This diagram will serve as a basis for illustrating how digital signatures fit into the overall security requirements of an application. The order of these steps may be different for some application architectures.

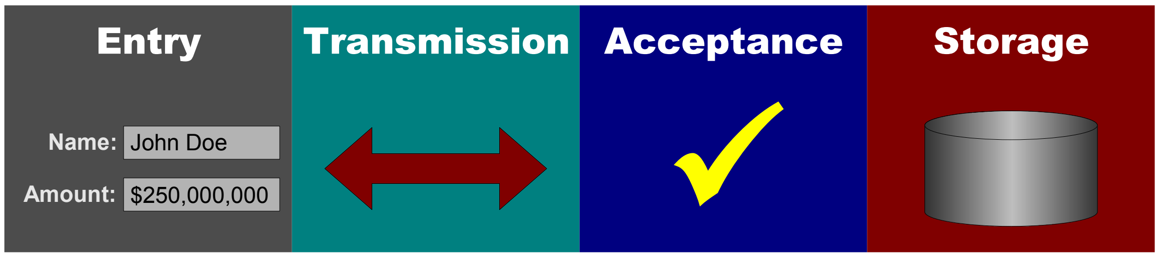


Figure 1: Four steps in a transaction

Step 1: Data Entry

Since transactions involve data, the data has to originate somewhere. This usually means that a user enters it on some sort of data entry screen. In this step, the application is probably concerned with data validation: ensuring that all required data fields are populated in a format that the application can understand. Applications may also want to prevent certain users from accessing certain data entry screens.

Step 2: Data Transmission

In many applications, transaction data is transferred across a network to a central application server and/or database server. Applications may need to ensure that the transaction data is not altered during transmission. Also, the transaction may include sensitive information such as credit card numbers or other private personal information. It is also likely that applications may require assurance that the data is being transmitted to the intended recipient. The popular SSL protocol satisfies these requirements for web-based applications. Virtual private networking (VPN) technologies can provide these services also.

Step 3: Acceptance

At some point in the process, the application or application server "accepts" the transaction. That is, the transaction meets all the requirements necessary to be processed. Accepting a transaction may involve several elements.

- **Data Validation**: all required fields are entered in a format that the application can understand,
- **Integrity**: the data has not been altered during transmission to the application or database server,
- **Authentication**: the identity of the user has been firmly established, and
- **Authorization**: the authenticated user has permission to perform this transaction.

Step 4: Storage

Since we are defining a transaction as an interaction between the user and the application that results in a change to the data stored in the database, the data must be stored. In many cases, a transaction requires that new data be written to the database. However, transactions might only change existing data. In either case, applications may need to ensure that the stored data is not changed, destroyed or viewed by malicious or unauthorized users. These attacks can often be prevented by a strong access control mechanism and a good backup plan.

2.2.2 Prevention vs. Proof

In the previous explanation, there is an element of transaction security that is missing. Let's examine what do we know at the acceptance stage (step 3).

- We know that all the required transaction data is entered in an acceptable format (validation).
- We know that the data has not been altered during transmission (integrity).
- We know that no one has viewed the data during transmission (privacy).
- We know the identity of the user performing the transaction (authentication).
- We know that the user has permission to perform the transaction (authorization).

It seems like all the major security requirements have been met. The problem is that we only know these things during the very brief period of time when the transaction is executed. Once the transaction is complete, this knowledge vanishes and cannot be re-established because it cannot be stored along with the transaction data. However, digital signatures allow some of this knowledge to be captured and stored.

However, digital signatures do not protect data in the same way as other cryptographic techniques. Digital signatures do not hide data from unauthorized viewers. This "confidentiality" is provided by data encryption. Digital signatures cannot prevent data from being modified by external hackers or malicious "insiders." This protection is provided by authentication and access control. Digital signatures simply allow an application to prove two things about the data they "protect":

- **Integrity**: the data has not been modified since it was signed, and
- **Origin**: the identity of the signer can be cryptographically proven.

There is a significant difference between ***preventing*** changes to application data and being able to ***prove*** that the data has not been changed. This may seem like a fine line, but how do you prove that your access control mechanisms have not been compromised? It is much easier to prove that a security violation has occurred than it is to prove that one hasn't. If attempts to defraud your organization are detected, then the hacker has not done a good enough job.

If the transaction data is digitally signed, applications that rely on that data can prove that it has not changed and that it came from an authorized user. So, although digital signatures cannot prevent fraud from being attempted, they prevent attempted fraud from succeeding by giving applications the ability to detect fraudulent transactions.

The digital signature itself is a separate piece of data that must be stored with the transaction to facilitate this proof. The fact that digital signature impacts the data storage requirements of the application is another reason why digital signature functionality requires a tighter integration with the application than other security technologies.

2.2.3 Paperless Business Processes

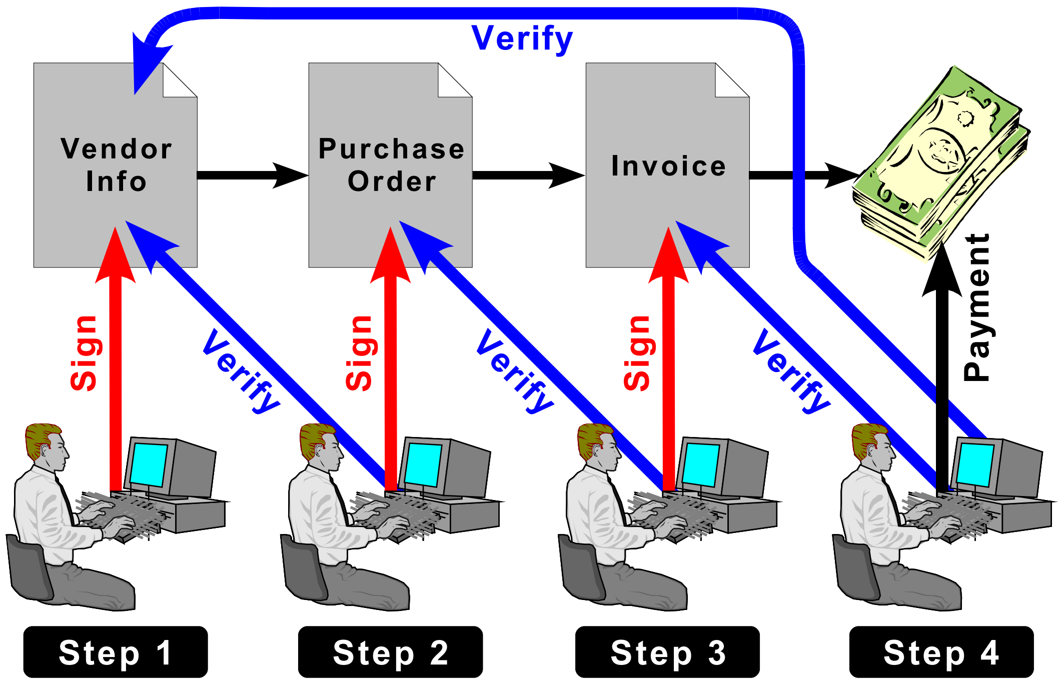


Figure 2: A typical paperless business process

Figure 2 shows how digital signatures are typically used to implement a paperless process. In each step, the users are using an application that allows them to view and modify data that is stored in a central database. Note that each time a “document” is created or modified within the application, it is digitally signed. Each time that data is used, its signature is verified. This allows the relying user to be confident that the data in the database is genuine and was originated by an authorized user. The application performs the signing and verifying automatically whenever a document is stored, modified, or retrieved from the database. This enforces the security policy and prevents users from inadvertently skipping these steps. Since the application must know when to sign documents, when to verify them, and what to do when either of these operations fail, digital signature must be an integral part of the application’s work flow logic.

2.3 Databases are Different

So far, we have discussed why digital signature is different than other security technologies. Relational database applications also have some very unique qualities. These unique qualities require a specialized approach to digital signature integration.

2.3.1 What is a Document?

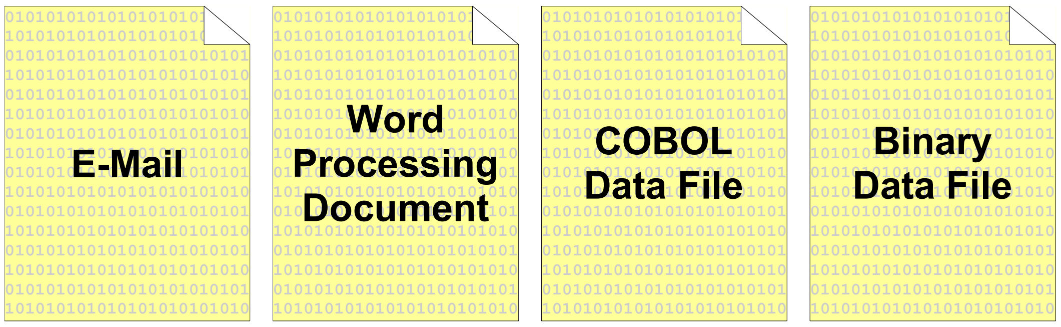


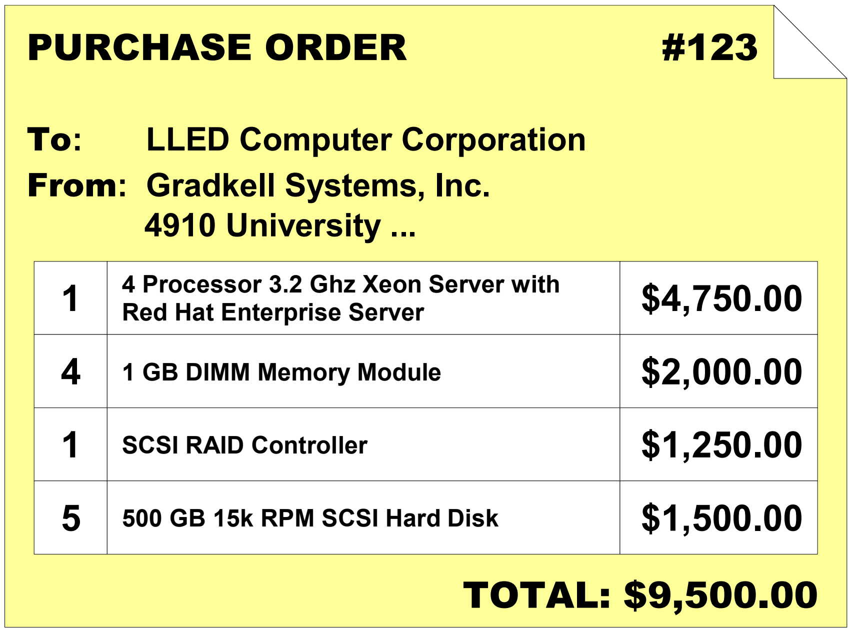
Figure 3: Types of documents

Earlier, we discussed digitally signed “transactions”. Often the term “document” is used to denote the data that is signed. Each type of digital signature solution seems to define a document differently. For example, email security products define a document as an email and its attachments. There are security products that digitally sign word processing documents or spread sheets. Other products digitally sign any type of file. Note that in each of these examples, even though a document may internally contain many discrete data elements, the document as a whole can be represented as a contiguous set of bytes.

Relational databases store their data much differently. Databases store structured data as opposed to unstructured data. This means that all of the data elements that compose a document must be known in advance before the first document is created. Databases use a concept called normalization, which allows large amounts of structured data to be stored and searched very efficiently. The data items in a document are stored in tables. Tables are composed of rows and columns. The columns define the name (e.g. “PRODUCT_NAME”, “INVOICE_NUMBER” or “PURCHASE_DATE”) and type (e.g., CHARACTER, NUMBER, and DATE, respectively) of each data element. A row in a table, called a “record”, contains the actual data values for each column in the table.

For our purposes a “document” is defined as the data in one or more rows from one or more columns of one or more tables in a relational database. That is, a document may span multiple database tables and may include only selected columns from those tables and may encompass more than one row per table. This sounds complex and it can be very complex. Databases are designed to efficiently handle large amounts of data that are related in complex ways.

Here is an example:



PURCHASE ORDER **#123**

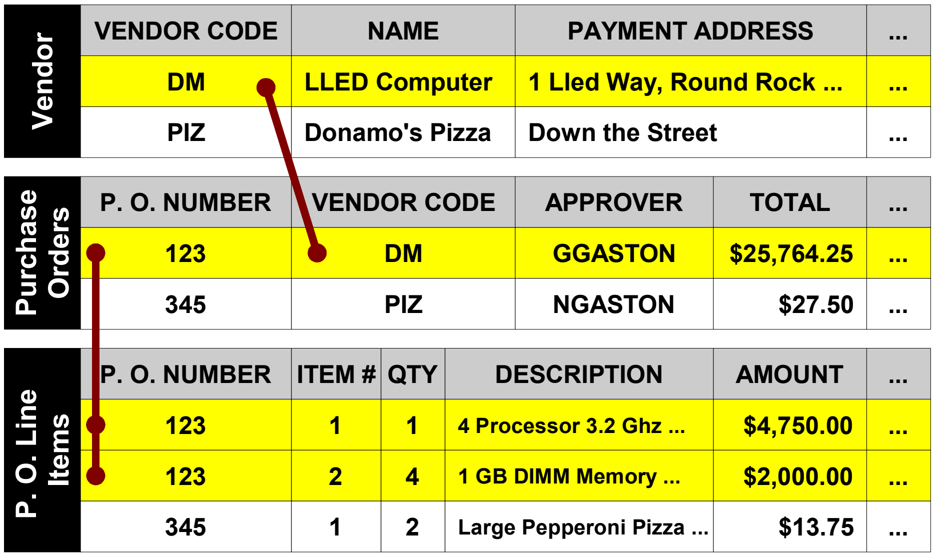
To: LLED Computer Corporation
From: Gradkell Systems, Inc.
4910 University ...

1	4 Processor 3.2 Ghz Xeon Server with Red Hat Enterprise Server	$4,750.00
4	1 GB DIMM Memory Module	$2,000.00
1	SCSI RAID Controller	$1,250.00
5	500 GB 15k RPM SCSI Hard Disk	$1,500.00

TOTAL: $9,500.00

Figure 4.: A database document printed or displayed by an application.

Figure 4 shows a document in a format that makes sense to people. It is a very simplified purchase order from Gradkell Systems, Inc. to a company named LLED Computer Corporation. A purchase order is usually identified by a purchase order number. This is purchase order #123. It has four line items. Each line item has a quantity, description, and amount. The purchase order also has a total amount. Figure 5 represents how purchase order documents might be stored in a database.



Vendor

VENDOR CODE	NAME	PAYMENT ADDRESS	...
DM	LLED Computer	1 Lled Way, Round Rock ...	...
PIZ	Donamo's Pizza	Down the Street	...

Purchase Orders

P. O. NUMBER	VENDOR CODE	APPROVER	TOTAL	...
123	DM	GGASTON	$25,764.25	...
345	PIZ	NGASTON	$27.50	...

P. O. Line Items

P. O. NUMBER	ITEM #	QTY	DESCRIPTION	AMOUNT	...
123	1	1	4 Processor 3.2 Ghz ...	$4,750.00	...
123	2	4	1 GB DIMM Memory ...	$2,000.00	...
345	1	2	Large Pepperoni Pizza ...	$13.75	...

Figure 5: A database document stored in the database. The rows in yellow pertain to purchase order #123.

Note that not all columns shown in Figure 5 are displayed in Figure 4. This is important because database applications may contain data that are used internally by the application but that are not important to the business process. Examples of such data are internal flags that mark a document's position in a work flow (e.g., it has been entered, but is pending approval). Is not usually necessary to sign this type of data because it is not really part of the document. This data is only used to move the document through a process. If it is signed, the signature will be invalidated when the data changes. So it is important to be able to choose which columns to include in the signature rather than having to sign the entire row.

Note that the data that pertains to purchase order #123 is not a contiguous set of bytes. It is intermingled with other purchase orders (e.g., #345, a pizza order). Since digital signature algorithms operate on a contiguous set of bytes, the data must be retrieved from the database and formatted into a contiguous string of characters. This must be done exactly the same way each time. The result must be bit for bit the same every time or the signature will not verify. This is because the digital signature operation is performed on a block of data. At the level in the process where the cryptography is applied, the contents of the data have no meaning. The signing process only sees the data as an ordered collection of bits. The signature verification process simply answers the questions "Is this the data that was signed?" and "Was it signed by the specified user?"

The exactness which data must be represented presents some special problems. Databases store numeric and date values in a special way and usually have a default format that is used to display these values. For example, if a date value was signed in the form “11:30 PM on 10 May 1999”, but was verified in the form “1999-05-10 23:30:00”, the signature will not verify because the data was changed. Actually, only the representation of the data has changed, but that representation was not bit for bit the same as when it was signed. The same is true of numeric data. The real number 47502.5 can also be represented as “$47,502.50”. This becomes an issue when the default format used by the database to represent numeric and date values can be changed by a database administrator. These problems can be avoided if the format of the data is explicitly specified when the data is retrieved from the database.

2.4 Basic Requirements for Digital Signature Integration

The following sections describe basic design goals for a generic database signing system.

No PKI knowledge required for application developers. Application developers should not have to become digital signature experts. Ideally they should not even need to understand what a digital signature is, other than that it is an operation that is performed on a certain document at a certain place in the business process. There are five application specific items that a generic database signature system cannot determine:

- What type of operation needs to be performed (e.g., signing or verification)
- What type of document is being signed or verified (e.g., a purchase request, an invoice, a time card, a leave request, a 401K participation form, etc.)
- Which specific document is being signed or verified (i.e., the “primary key” values that uniquely identify a single document)
- When in the business process to perform digital signing or verification
- What to do if an error occurs during signing or verification

All of these items are known by the application developer and are similar to the types of information required by other operations in the application. For example, an application developer has to know that “purchase request number 123 needs to be signed when the user presses the submit button”. Of course, the actual process is much more complex, but the application developer does not need to know the other details such as which columns in which tables are signed or where the signature data is stored.

Does not require modification to the existing database structure. If the digital signature system is to be application independent, it should not directly rely on the database structure of a certain application. Adding new tables should not be problem, however.

Allows the data that is signed to be specified. Since databases do not store their data as contiguous sets of byte, the data items that compose a document or transaction must be gathered from the database. The data that is signed has to be exactly the same when it is verified as when it was signed. Since we want this system to be very easy to integrate, we do not want to burden application developer with this task. Since the digital signature will be performing the data-gathering step, it must allow the data (tables and columns) to be specified. This specification should include information that defines how each data item is to be formatted (e.g., “1:00 PM” or “13:00”). The specification should also be able to represent the “primary keys” of the document and the complex ways that the underlying tables are related to each other.

Scalable and does not introduce a single point of failure. The database server and the application server are all required by the application. The PKI adds a directory server. The digital signature system should not introduce any additional servers that could become a bottleneck or cause application processing to stop.

Signature storage overhead should be as small as possible. Database environments offer great advantages when it comes to the efficient storage of data. The de facto standard format for digital signature storage is PKCS #7, the cryptographic message syntax standard. This standard defines a data structure for cryptographic messages such as signed documents.

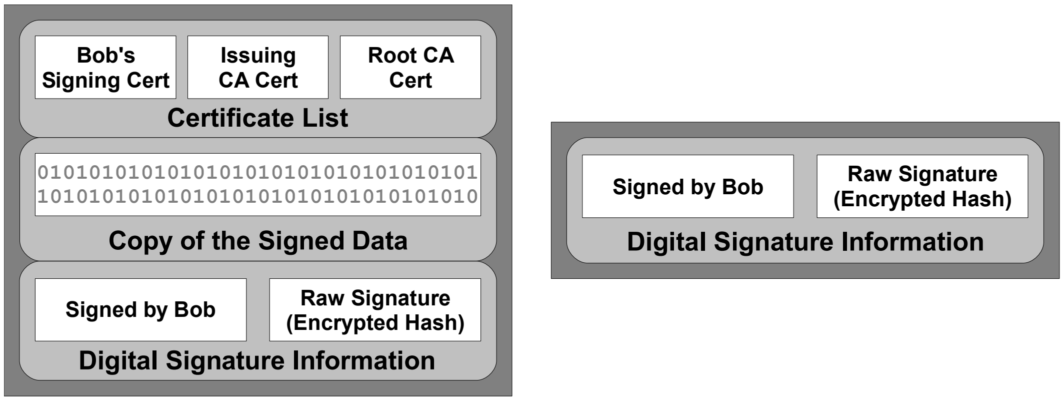


Figure 6: A typical PKCS #7 signed data message verses one that has been optimized for storage in a database.

Most of the fields are optional, but a typical signed data message includes the signer's certificate, the other CA certificates in the "chain" and a copy of the data that was signed. Essentially a PKCS #7 signed data message is a large "de-normalized" chunk of binary data. Since the database is a central data repository that is shared by the signer and the verifier, the certificates and the data do not need to be stored with each signed document. Since this data is being stored in a database, it can be "normalized". The certificates can be stored only once and linked to the signed document via database relationships. A single certificate is about 600 to 1000 bytes in size. A typical PKCS #7 message contains about 3 certificates. The data portion, which is of indeterminate length, can be also removed from the PKCS #7 message because the data is already stored in the database and does not need to be stored again. As Figure 6 shows, the normalization of the signature information greatly reduces the amount of signature storage overhead required by the digital signature system. The "optimized" PKCS #7 is about 300 bytes long vs. over 3000 bytes (assuming 1024 bytes of data) for the typical case. Storing less data per document also improves also improves performance because less data has to traverse slow network connections.

2.4.1 Abstracting the Digital Signature Process

Digital signature integration can be viewed as "bolting on" digital signature functionality onto an existing application. The actual cryptographic operations and interaction with PKI components are performed by low-level cryptographic toolkits. The "digital signature bolt on" is a program library that knows how to interact with both the database and the cryptographic toolkit.

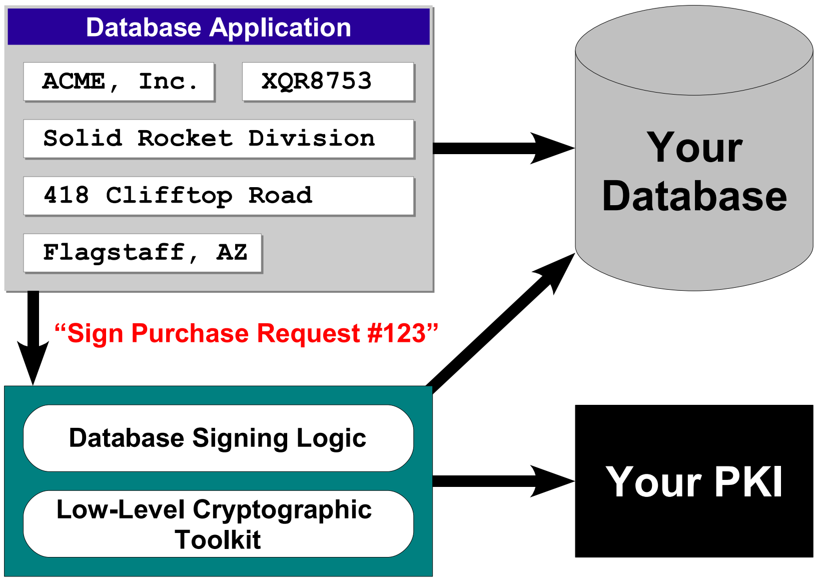


Figure 7: The process of signing a database "document" is standardized and removed from the application logic.

In Figure 7, the low-level cryptographic toolkit only knows how to sign raw data. It does not know how to gather it from the database or how to store signature information in the database. The database signing logic knows how to retrieve the purchase request data from the database and how to use the cryptographic toolkit to sign the data. It also handles formatting the signature data in a way that is optimal for storage in the relational database environment.

Essentially, the process of digitally signing data in a database is standardized and abstracted from the application so that the application developer does not have to know anything about it. The developer provides just enough information to get process started. The rest is handled automatically.

2.5 Summary

In this section we have discussed some of the unique qualities of both digital signatures and relational databases. Digital signatures are different because they require that data be stored to support signature verification. Relational databases are different because they store data in a very unique way. These two differences work together to make integrating digital signatures into relational database applications a complex and tedious task. The cost and risk of this crucial integration step has hindered the use of digital signatures in many applications. Until recently, there have been no digital signature products specifically designed for the database environment. DBsign leverages the cryptographic and security expertise of specially trained third-party developers to drastically reduce the cost and risk associated with trying to tackle complex, highly technical integration projects in house.

3.0 DBsign Concepts

3.1 The Purpose of DBsign

DBsign is a digital signature system designed specifically to provide provable data integrity and technical non-repudiation for data stored in relational databases. DBsign includes both a Software Development Kit (SDK) and a set of graphical administration tools that work together to make the integration of digital signatures into database driven applications a quick and easy process.

The DBsign Software Development Kit includes a simple, high-level application programming interface (API) that minimizes changes to existing application code. No specialized cryptographic or digital signature knowledge is required of developers or users.

The DBsign Administration Tools control the security and configuration parameters under which DBsign operates. The tools centrally control and manage every aspect of the digital signature system.

3.2 Feature Overview

Feature	Description
Easy to Integrate	Since DBsign was designed exclusively for database applications, integration is fast, easy and straightforward.
Easy to Maintain	DBsign includes graphical administration tools that allow secure, centralized, point-and-click management of every aspect of the digital signature system.
Easy to Deploy	DBsign is easily deployed to hundreds of thousands of users. All configuration is centralized and digitally signed (e.g., there are no client side registry settings, environment variables or configuration files to maintain). The DBsign Universal Web Signer client-side software does not require installation, but is automatically downloaded as part of the web page much like an image or a flash animation.
Application Independent	DBsign is not tied to any certain application architecture or development environment. It can work in many types of applications including client/server and web-based architectures.
Database Independent	DBsign works with all major relational database systems including Oracle, DB2, SQL Server and many others.
PKI Independent	DBsign is interoperable with all major PKI products and services including RSA Keon, Verisign OnSite, Entrust PKI, Netscape CMS, Baltimore UniCert and other standards-based PKIs.

Platform Independent	DBsign works on most major 32-bit and 64-bit desktop and server operating systems such as Microsoft Windows, Apple Mac OS X, Sun Solaris, Linux, and others.
Based on open database and cryptographic standards	DBsign achieves a high-level of interoperability by carefully adhering to open standards.
Advanced PKI Features	DBsign's shared, multi-level Certificate and CRL cache/archive supports long-lived documents and significantly reduces access to LDAP directories. This increases application performance and provides a resiliency not found in other digital signature implementations.
Advanced Cryptographic Features	DBsign supports advanced hardware cryptography such as smart cards, USB tokens, high-speed crypto-accelerators, etc.
Adheres to U.S. Government Cryptographic and Security Standards	DBsign supports a variety FIPS 140 validated cryptographic modules and has passed 100% of DoD's PKI security and interoperability tests (performed by the Joint Interoperability Test Command). DBsign has also been validated under the NIAP CCEVS program confirming that DBsign is a secure product that does what it says it will do.
Based on 20 years of digital signature experience	Gradkell Systems Inc. has over a decade of experience in digital signature integration and relational database development. During this time, GSI developed the only electronic system sanctioned by the U. S. General Accounting Office as a legally binding signature.

Figure 8: DBsign Feature Overview

3.3 Basic Architecture

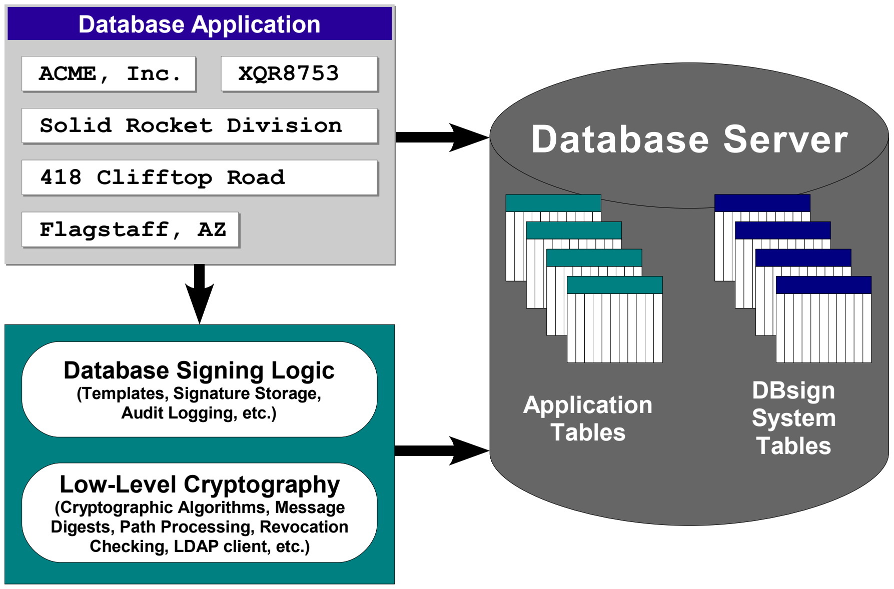


Figure 8: Basic DBsign Architecture

Figure 8 is a high-level, conceptual diagram that show how DBsign fits into relational database applications. The diagram shows three major parts of DBsign: database signing logic, low-level cryptography, and the DBsign system tables. Note that the database signing logic and low-level cryptography may execute on different tiers depending on the application architecture.

DBsign's database signing logic encapsulates all the non-cryptographic functionality required to digitally sign application data. This functionality includes querying the application tables (via "Signature Templates"), storing digital signatures and other supporting data in the databases, and maintaining an audit log of all digital signature related operations. Actual database-level access is provided through DBsign Query Modules.

The low-level cryptographic operations perform the actual cryptographic operations associated with signing data or verifying signatures. These cryptographic operations also involve message digests, public-key cryptographic algorithms, certificate/CRL caching, certificate path building and validation, and interacting with the public-key infrastructure to determine certificate revocation status. The low-level cryptographic functionality is provided by DBsign's cryptographic subsystem which can use a variety of cryptographic implementations.

The DBsign System Tables contain configuration data that controls how both the database signing logic and the low-level cryptography components operate. This configuration data is stored in the database and centrally managed via the DBsign Administration Tools.

3.4 The DBsign System Tables

DBsign does not require any modification to an application's database structure. However, DBsign does require some of its own tables to be installed. These DBsign specific tables are called the DBsign "system tables". These tables are created and populated by the DBsign Administration Tools. The system tables are usually stored in a separate schema (or user) named "DBS". All the DBsign table names are prefixed by "DBS_" so that they will easily integrate into applications or database products that do not use schema.

DBsign has the ability to fully qualify all queries made to the DBsign system tables. For example, if the DBsign tables are stored in a schema named "DBS", then a query will be made on "DBS.DBS_TMPL_VERSIONS" instead of just "DBS_TMPL_VERSIONS". This means that you do not need to make synonyms or aliases for the DBsign system tables. This is also an important security feature because it prevents DBsign from using "rogue" copies of the DBsign system tables that malicious users may have copied into their own schema and modified.

Much of the data in the DBsign system tables is digitally signed. DBsign verifies the signatures on this data before using it. This ensures that DBsign is not basing its security decisions on data that may have been maliciously altered. The non-repudiation feature of digital signatures also enforces personal accountability for changes made to the security sensitive configuration data contained in the DBsign system tables.

For more information about the required access control permissions to the DBsign system tables, see Appendix B.

3.5 DBsign Signature Templates

All relational databases that use the Structured Query Language (SQL) have at least one thing in common: the data they contain can be extracted (i.e., queried) and stored (i.e., inserted or updated) in a standardized way. This is not true of other types of data such as word processing documents, email, COBOL data files, etc. DBsign's concept of "digital signature templates" takes advantage of this unique quality of relational database technology.

Essentially, DBsign standardizes the process of digitally signing and verifying data stored in relational databases. That is, DBsign goes through exactly the same process for each type of document that it signs or verifies. Digital signature templates are perhaps the most important part of DBsign's integration into applications because they define how DBsign will operate on data that is unique to the host application.

DBsign signature templates have the following qualities:

- They specify which data elements to sign or verify (i.e., which columns in which tables).
- They specify which data values uniquely identify the data to be signed or verified (i.e., which columns in which tables represent the "primary keys" for the data).[1]
- They specify the data format for each data item being retrieved (data must be represented in the same format when it is signed and verified).

1 Primary key constraints, unique indexes, or referential integrity constraints are not required to use DBsign. However, you must be able to uniquely identify the rows that pertain to a certain "document" by the specification of data values for one or more database columns.

- They specify how data is related so that complex database relationships can be represented (i.e., one-to-one, one-to-many, many-to-many, etc.).
- They specify where to store the digital signature information in the database (i.e., which table).
- They are versionable so that DBsign can adjust to changes in your application's data requirements while preserving the integrity of previously signed data.

Signature templates are created using the DBsign Template Designer Administration Tool. This graphical tool makes it easy to create, test and troubleshoot signature templates. The resulting signature template is stored in the DBsign system tables and is digitally signed to protect it from modification. Whenever DBsign loads the template from the database, the template's signature is verified. DBsign "caches" signature templates in client-side memory so that they are only loaded and verified once per DBsign session.

An "item" in a DBsign signature template specifies the schema, table and column of a piece of data in the database. DBsign signature templates have two types of "items": primary key items, and data items. Primary key items are exactly like data items except that they are used to uniquely identify a certain set of rows in the tables included in the template. When signing data with DBsign, actual data values are specified for the primary key items. This allows DBsign to retrieve the data for a specific document. Values must be supplied for all primary key items defined in the template.

All template items also have a "data type" which determines how to represent the data item as a string (for example a DATETIME field may be represented as "2000-12-15 13:45:01", and a numeric field used for dollar amounts may be represented as "$5,267,278.47"). Applications may define as many data types as they need. A template item's data type also contains a field that specifies how to convert the string representation of the item data back into a native database type. This is used to convert string values specified for primary key items so that the values can occur in the WHERE clause of a select statement.[2] Data type information is configured using the DBsign Data Type Manager Administration Tool.

To represent complex database relationships, DBsign signature templates can be related to each other just like the underlying database tables are related to each other. In DBsign, these relationships are called "template dependencies". For example, to represent a one-to-many relationship between two tables A and B, use the DBsign

2 Several templates may share data type information but the data type information is included in the digital signature on each template that uses it. Therefore, if you change a data type specification, you must re-sign each template that uses the changed data type specification.

Template Designer create a template for each table. Then, make template A dependent on template B by specifying the columns in table A that migrate to table B. When your application instructs DBsign to sign a document defined by template A, DBsign "executes" template A. Then, for each record returned by template A, template B is executed using the values for the columns that migrate from A to B. These dependency relationships can be as complex and as deeply nested as required. Circular dependencies are not supported (e.g., template B cannot be dependent on template A if template A is already dependent on template B).

DBsign signature templates are also versionable. There is always at least one version of a template. Each template version is time stamped with an "effective date". DBsign always uses the most recent version of the template when signing. When the digital signature is stored in the database, the date and time of signing is stored as part of the signature information.[3] When verifying a signature, DBsign uses the document's sign date and the template's effective date to determine the version of the template that was used to sign the template. DBsign includes the date and time of signing as part of the data that is signed. This prevents malicious users from "time shifting" documents in an attempt to bypass DBsign's certificate expiration or revocation checks.

When DBsign signs or verifies a document, the template is used to collect the data in the document. The Extensible Markup Language (XML) is used to format the data into a string for signing. The XML document type definition (DTD) for this format is freely available.

3.6 The DBsign Audit Logging System

In order to enhance security and to facilitate troubleshooting, DBsign includes a comprehensive, but configurable logging feature. Most security related DBsign operations generate log entries. The audit log makes it possible to determine who did what and when.

The DBsign audit log is stored in the database in the DBS_LOG_*xxx* tables. These tables are linked by a log entry number (LOG_NO) that is generated internally by DBsign.[4] There are four types of DBsign log entries: login (1), logoff (2), sign (4), and verify (8).

3 DBsign does not rely on the client workstation as a time source. Since secure timeservers are still rare and not part of "typical" PKI environments, DBsign uses the database server as a time source. Support for secure timeservers may be added to future releases. The database server is assumed to be in local time. GMT conversion is based on the time zone setting of the client workstation. Future releases of DBsign will include the ability to specify the GMT offset of the time returned by the database server.

4 Since database products differ widely in how they generate globally unique identifiers (or "sequences"), DBsign implements its own sequence generator using the DBS_SEQUENCES table.

The logging system can “filter” which types of log entries are included in the logs. The DBsign Log Settings Tool is used to set up this filter. These settings allow each type of log entry to be filtered based on the success or failure of the operation.

The audit logging system requires its own database connection. This is to ensure that the logged data can be committed to the database without effecting the application’s current transaction. It also ensures that the logged data will not get rolled back by the application should the application abort a transaction.

The DBsign audit logging system can be used to research signature failures. When a signature fails to verify because data was changed, it is important to be able to determine which data items were changed. DBsign accomplishes this by logging a copy of the data (in a highly compressed form) whenever data is signed. This allows DBsign to present a “before and after” picture of the data and to identify the offending data elements.

The audit logging feature is required for normal operation. The minimal data that can be logged are successful signature generations and failed signature verifications.

3.7 DBsign Database Access

The DBsign Server uses the two most popular open standard database APIs to access databases: Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC). Using open standard database interfaces in conjunction with using an interoperable subset of ANSI SQL syntax allows DBsign to work with most popular database products.

3.8 DBsign Cryptographic Subsystem

DBsign assumes the presence of an existing PKI and assumes that user’s have already received certificates and private keys. Since DBsign does not generate certificate requests, renewal requests, or revocation requests, the main points of interoperability with the PKI relate to the certificate repository and certificate revocation status methods. The user’s certificate and private keys may be stored in software as well has hardware devices such as smart cards, USB tokens, PC Cards, or high-speed cryptographic accelerators.

DBsign supports several cryptographic modules that have undergone stringent testing and validation under the National Institute of Standards and Technology's (NIST) FIPS 140 cryptographic module validation program.[5]

The DBsign Cryptographic Subsystem has a unique feature that allows it to cache certificates and CRLs in memory as well as in the database. The shared database cache serves as a certificate and CRL archive. This archive allows DBsign to support documents with a lifetime much longer than the certificate expiration period. For example, certificates usually expire in just a few years. CRLs, to reduce their size, typically only contain revocation information for current, unexpired certificates. In many applications signatures must be verifiable for decades or more. When verifying signature on documents signed with certificates that have since expired, revocation status cannot be performed using the current CRL. DBsign allows revocation status to be performed by using the archived CRL.

The DBsign Cryptographic Subsystem also supports very efficient storage of digital signature information. DBsign uses the standard Cryptographic Message Syntax (PKCS #7) standard to represent a digital signature. However, most of the items stored in a PKCS #7 signed data message are optional. By default, DBsign includes only the fields that are required to verify the digital signature. The resulting digital signature is very small in size because it does not contain any information that is redundant in the context of the broader application (such as certificates, CRLs or a copy data that was signed). DBsign stores this data elsewhere in the database in a normalized fashion that is optimal for relational database applications. DBsign can also include these fields in the signature including the signer's certificate, the full certificate chain, and the data that was signed.

The DBsign Cryptographic Subsystem supports operating system cryptographic services such as Microsoft CryptoAPI and Apple Max OS X KeyChains for certificate and private key storage. This makes it easy for DBsign to share certificates with other applications such as Microsoft Outlook and Internet Explorer. Certificates and private keys can also be shared with Mozilla Firefox via DBsign support of NSS. DBsign can also use keys stored in PKCS #12 files.

5 Please note that although DBsign uses cryptography, it is not itself a cryptographic module as defined by FIPS 140 and thus is not subject to evaluation under the FIPS 140 standard. However, the cryptographic modules that ship with DBsign *have* been validated under this program. To learn more about FIPS 140 and NIST's cryptographic module validation program, please see their web site at http://csrc.nist.gov/cryptval.

3.9 The DBsign User Policy Feature

DBsign includes an optional feature that allows applications to restrict who may use DBsign. Specifically, this feature allows the application to restrict:

- Who may digitally sign documents
- Which certificates may be used to sign documents
- What types of documents individual users may sign

The DBsign user policy system is disabled by default because users must be defined using the DBsign User Manager Administration Tool. Since the user policy system is optional, it is always disabled within the DBsign Administration Tools. Applications may enable this feature with the DBS_RequireUsers() and DBS_SetuserName() API calls.

Each DBsign user account has the following attributes:

● **User Name**:	A name that uniquely identifies the user to DBsign
● **First Name**:	The user's first name. Used for informational purposes only
● **Last Name**:	The user's last name. Used for informational purposes only
● **Active Flag**:	If set to "N", the user may not login to DBsign or sign documents. Used to disable a DBsign user account
● **Certificates**:	A list of certificates with which the user is permitted to sign documents

Each certificate has the following attributes:

● **Security Level**:	The DBsign security level of this certificate
● **Description**:	A short description of this certificate's intended use
● **Notary Flag**:	A flag indicating that signatures are used for data integrity purposes only.

Each user account may have multiple certificates associated with it. If there is more than one certificate associated with the user, DBsign will prompt the user for which one to use. A list of certificates will be displayed. This display includes the certificate description text so the user can easily determine the correct signing certificate to select. If there is only one certificate associated with a user, DBsign will not prompt the user.

Each certificate associated with the user account also includes a "security level". Each DBsign digital signature template also includes security level. When the DBsign User Policy feature is enabled, DBsign will not allow a user to sign a template if the security level of the template is higher than the security level of the user’s certificate. This allows some control over who is permitted to sign certain types of documents.

This feature is typically used in environments where a user has more than one certificate and the application does not want to require the user to select the correct certificate. It is also use to control which users have “signature authority” in the application.

Another use of the DBsign User Feature is security/cost optimization. Some organizations may wish to deploy cryptographic tokens (e.g. smart cards or USB tokens) to certain key employees, while allowing the rest of the staff to use software cryptography. The security-level feature DBsign users and templates can specify that certain types of transactions be signed with high security devices while allowing most transactions to be signed with software cryptography.