NexusDB Security Pack

An overview of the NexusDB Security Pack and comparison of the various encryption engine provided within

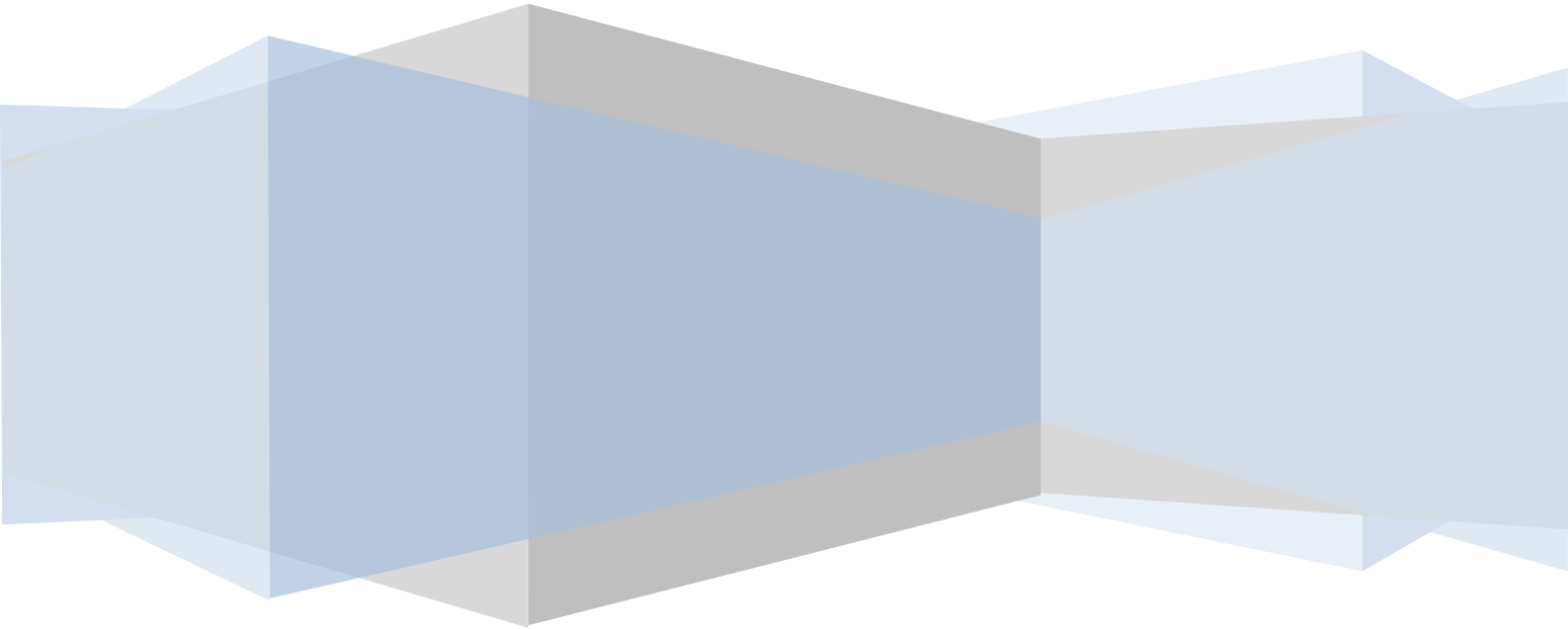

Table of Contents

Overview

The NexusDB Security Pack is a new add-on for NexusDB which provides a number of highly secure industry standard data encryption engines. These engines are designed, implemented and documented under contract by highly regarded data security consultant Henrick Hellström of StreamSec HB (http://www.streamsec.com), a Swedish company specializing in the development of highly secure data encryption.

The Security Pack is available in three versions: Standard AesCcm, Standard XtsAes, and Professional. The Security Pack is designed to work as an extension engine in NexusDB, thus making integration and maintenance of the security pack into existing systems as simple as possible. The packs are delivered as Developer Edition installers with full source code, and the installers will download server and Enterprise Manager binaries with the engines compiled in.

- Standard AesCcm Pack comes with 128- and 256-bit versions of the AesCcm encryption engine. This encryption engine is suitable for

Feature matrix:

Engine	Standard AesCcm	Standard XtsAes	Professional
AES CCM 128	X		X
AES CCM 128 (assembler optimized)			X
XTS AES 128		X	X
AES GCM 256			X
AES CCM 256	X		X
XCB AES 256			X
XCB AES 256		X	X
SHACAL CBC HMAC 256			X
AES CBC MAC 128			X

From a database specific point of view a simple restructure of the table with the Encryption engine identifier set to one of the new engines will secure the database.

Technical comparison

Scheme	Identifier	Tier	Key size	Confidentiality mode	Integrity mode	Random generator	Password processing	Key wrap	Supports table checksum	Overhead
Blowfish RC4	nx1xDefault	nxdb	128 bits	CBC*	XOR checksum with ECB and RC4 whitening	non standard	Blowfish and RC4 key schedule	Blowfish RC4	No	• 48 bytes per table • 16 bytes per block
AES CCM 128	nxsAesCcm128	STD	128 bits	CTR	CBC MAC	ANSI X9.31 AES	PKCS#5 PBKDF-2 with AES based PRF from NIST SP800-90	AES CCM 128	No	• 64 bytes per table • 32 bytes per block
XTS AES 128	nxsXtsAes128	STD	128 bits	XEX	none - 128 bit granularity	ANSI X9.31 AES	PKCS#5 PBKDF-2 with AES based PRF from NIST SP800-90	AES CBC and AES CBC MAC with independent keys	No	• 96 bytes per table • 0 bytes per block
AES GCM 256	nxsAesGcm256	PRO	256 bits	GCTR	GHASH	NIST SP800–90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	AES GCM 256	No	• 80 bytes per table • 32 bytes per block

AES CCM 256	nxsAesCcm256	PRO	256 bits	CTR	CBC MAC	NIST SP800−90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	AES CCM 256	No	• 80 bytes per table • 32 bytes per block
XCB AES 256	nxsXcbAes256	PRO	256 bits	XCB	none - full table block granularity	NIST SP800−90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	AES CBC and AES CBC MAC with independent keys	No	• 80 bytes per table • 0 bytes per block
XCB AES 256	nxsXcbAes256Red	PRO	256 bits	XCB	Redundancy check	NIST SP800−90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	AES CBC and AES CBC MAC with independent keys	Yes	• 96 bytes per table • 32 bytes per block
SHACAL CBC HMAC 256	nxsShacalCbcHmac	PRO	256 bits	CBC	HMAC SHA 256	NIST SP800−90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	SHACAL CBC HMAC 256	Yes	• 160 bytes per table • 64 bytes per block
AES CBC MAC 128	nxsAuthAesCbcMac128	PRO	128 bits	none	AES CBC MAC 128	NIST SP800−90 AES CTR	PKCS#5 PBKDF-2 with HMAC-SHA256	SHACAL CBC HMAC 256	No	• 64 bytes per table • 16 bytes per block

* with RC4 whitening

Performance Comparison

Scheme	Identfier	Encrypt 4kb blocks	Encrypt 64kb blocks
Blowfish RC4	nx1xDefault	23 MB/s	25 MB/s
AES CCM 128 (Std)	nxsAesCcm128	9.9 MB/s	10 MB/s
AES CCM 128 (Pro)	nxsAesCcm128	62 MB/s	65 MB/s
XTS AES 128	nxsXtsAes128	20 MB/s	20 MB/s
AES GCM 256	nxsAesGcm256	34 MB/s	35 MB/s
AES CCM 256	nxsAesCcm256	50 MB/s	52 MB/s
XCB AES 256	nxsXcbAes256	22 MB/s	22 MB/s
XCB AES 256	nxsXcbAes256Red	21 MB/s	22 MB/s
SHACAL CBC HMAC 256	nxsShacalCbcHmac256	36 MB/s	39 MB/s
AES CBC MAC 128	nxsAuthAesCbcMac128	115 MB/s	117 MB/s

Practical and Certification Comparison

Scheme	System requirements	Use for...	Certified for...	DO NOT use for...	Notes
All	• The adversary MUST NOT have access to the nxserver.exe process while running. • Disable virtual memory and the page file for the computer where nxserver.exe runs. • Do not hibernate. • Use RAM memory modules that are cleared on power failure. Use hardware with counter measures (or other equivalent means) against cold boot attacks. • Client users must manage table passwords securely.	• Protecting data server side.		• Do not use for DRM. • Do not use for preventing the software user from modifying data provided by the manufacturer and used by the software.	• All encryption engines are to different extent vulnerable to traffic analysis, where the adversary observes disk activity and correlates it to application events. • All encryption engines are vulnerable to replay attacks, where the adversary rolls back an entire table block to an older version of that table block.
Blowfish RC4	• Due to potential bugs in BIOS and HAL that affect QueryPerformanceCounter on multi core CPU systems or multi CPU systems, this encryption engine should only be used on older single core CPU systems.	Generic low end security for small tables. • Up to 128 bit confidentiality for small read only tables that were filled with data relatively quickly on a single run. • Lower confidentiality and no guaranteed authentication in all other cases, due to potential IV collisions. (Partial information might leak.) • Up to 64 bit integrity and authentication.	Not certified	• Do not use in any scenario where 128 bit confidentiality, integrity and authentication is required. • Do not use if nxserver.exe is running on modern hardware.	
AES CCM 128 **Standard**	• nxserver.exe must always work against the latest version of each table file. Live backups from older versions of a table file are permitted. Some integrity of the physical storage has to be	A nxserver.exe running on a machine with a RAID drive or one that might take file system backups of the table files (although not use them for restoration).	Live backup - write once	• Do not use if nxserver.exe is running in a virtual machine, in a cloud, or in any other scenario	• Only 20 bit confidentiality if the adversary is able to roll back the physical table files without detection. There is a 0.5 risk of complete loss of confidentiality of at least

	provided using external means.	• 128 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware), even if the attacker at the same time gets access to multiple historical versions of the same table files (e.g. file system backups). • 128 bit authentication of stored data when recovering from a sudden detected security breach. (I.e. ability to detect if an already detected intruder planted unauthentic table blocks in the table files.) • 128 bit integrity of stored data when recovering from a sudden detected security breach, provided that a nxdb live backup is used for restoration. If file system backups are used the intruder might substitute different historical versions of the same table blocks without detection.		where backup and restoration might happen transparently.	two table blocks after 2^{20} table block re-encodings. • Only certified for write-once backups, but might be used for other purposes if extra care is taken.
XTS AES 128 **Standard**	• Any security breach must be detected immediately using external means. Use other software for malware detection and for preventing intrusion. • Live backups must be stored securely.	A nxserver.exe running on a PC or laptop, ideally with a file system without caching, such as FAT32. • 128 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware), provided that no file system backups are made.	Disk encryption for confientiality in case of a sudden detectable security breach (e.g. theft of hardware)	• Do not use if the adversary might get access to multiple historical versions of the same table file. • Do not use if there is any chance that an adversary might modify a table file the nxserver.exe has to use.	• Limited confidentiality if the adversary gets access to multiple historical versions of the same table files. Partial information might leak. • No integrity check or authentication.
AES GCM 256 **Professional**	• nxserver.exe must always work against the latest version of each table file. Live backups from older versions of a table file are permitted. Some integrity of the	A nxserver.exe running on a machine with a RAID drive or one that might take file system backups of the table files (although not use them for	Live backup - write once	• Do not use if nxserver.exe is running in a virtual machine, in a cloud, or in any	• Only 16 bit confidentiality if the adversary is able to roll back the physical table files without detection. There is a 0.5 risk of complete loss of

	physical storage has to be provided using external means.	restoration). • 256 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware), even if the attacker at the same time gets access to multiple historical versions of the same table files (e.g. file system backups). • 128 bit authentication of stored data when recovering from a sudden detected security breach. (I.e. ability to detect if an already detected intruder planted unauthentic table blocks in the table files.) • 128 bit integrity of stored data when recovering from a sudden detected security breach, provided that a nxdb live backup is used for restoration. If file system backups are used the intruder might substitute different historical versions of the same table blocks without detection.		other scenario where backup and restoration might happen transparently.	confidentiality of at least two table blocks after 2^{16} table block re-encodings. • Only certified for write-once backups, but might be used for other purposes if extra care is taken.
AES CCM 256 **Professional**	• nxserver.exe must always work against the latest version of each table file. Live backups from older versions of a table file are permitted. Some integrity of the physical storage has to be provided using external means.	A nxserver.exe running on a machine with a RAID drive or one that might take file system backups of the table files (although not use them for restoration). • 256 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware), even if the attacker at the same time gets access to multiple historical versions of the same table files (e.g. file system backups). • 128 bit authentication of stored data when recovering from a sudden detected security breach. (I.e. ability	Live backup - write once	• Do not use if nxserver.exe is running in a virtual machine, in a cloud, or in any other scenario where backup and restoration might happen transparently.	• Only 20 bit confidentiality if the adversary is able to roll back the physical table files without detection. There is a 0.5 risk of complete loss of confidentiality of at least two table blocks after 2^{20} table block re-encodings. • Only certified for write-once backups, but might be used for other purposes if extra care is taken.

		to detect if an already detected intruder planted unauthentic table blocks in the table files.) • 128 bit integrity of stored data when recovering from a sudden detected security breach, provided that a nxdb live backup is used for restoration. If file system backups are used the intruder might substitute different historical versions of the same table blocks without detection.			
XCB AES 256 **Professional**	• Any security breach must be detected using external means. Use other software for malware detection and for preventing intrusion. • Live backups must be stored securely.	Secure systems with explicit intrusion detection but not disk level encryption, or secure systems with or without disk level encryption that grant read access to not completely trusted entities, or a PC or laptop with NTFS. • 256 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware).	Disk encryption for confientiality in case of a sudden detectable security breach (e.g. theft of hardware)	• Do not use if there is any chance that an adversary might modify a table file the nxserver.exe has to use.	• No integrity check or authentication.
XCB AES 256 (Redundancy) **Professional**	• Some integrity of the physical storage has to be provided using external means.	Secure systems with explicit intrusion detection but not disk level encryption, or secure systems with or without disk level encryption that grant read access to not completely trusted entities, or a PC or laptop with NTFS. • 256 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware). • 256 bit confidentiality for individual table blocks.	Disk encryption for confientiality in case of a sudden detectable security breach (e.g. theft of hardware)		• Table checksum prevents replay attacks, provided that the system tables are stored securely.
SHACAL CBC HMAC 256 **Professional**	• Some integrity of the physical storage has to be provided using	Secure systems with explicit intrusion detection but not disk level encryption, or secure systems with or without disk			• A 256 bit block size means that there is less than 2^{-152} chance of a state collision

	external means.	level encryption that grant read access to not completely trusted entities, or a PC or laptop with NTFS. • 256 bit confidentiality in case of a sudden detectable security breach (e.g. theft of hardware). • 256 bit confidentiality for individual table blocks.			in CBC if the largest table block size is used and the maximum amount of table blocks are encoded. If historical data is available, there has to be 2^{24} versions available of such maxized tables before the chance of a collision reaches 2^{-128}. • Table checksum prevents replay attacks, provided that the system tables are stored securely.
AES CBC MAC 128		• Prevent modification of non-confidential data.			• Integrity and authentication only - not a confidentiality mode.

Usage

Cryptography has to do with risk management, but in several respects very differently from how financial risks are managed. It is important to fully understand the difference before deciding on whether certain security measures are necessary.

Illustration 1
Suppose you have locked yourself out of your house. It's winter and you stand outside freezing, ill dressed with no mobile phone and - obviously - no keys. You are determined to get inside, fast. Do you stare at the lock of your front door and give up? No, you go around back and search for a way in and won't give up until you have found one.

When dealing with cryptography it is beneficial to picture the adversary as yourself, locked outside and trying to get in. The most common mistake done by developers is to think that it is "unlikely" that the attacker would attempt to get in one way but not another. Just like you in the example wouldn't stop until you're safe inside, the attacker won't stop just because the most obvious way in happens to be closed.

Does that mean it is wrong to even try to assess the risk your home will be burglarized? Should all residential areas look like Fort Knox and all houses have state of the art security systems? Obviously, no. Economics does play a role in security.

- Firstly, most people are not concerned about burglars who would target their home specifically. Similarily, some computer attacks follow the same logic, such as attackers who spread malware and get enough credit card information and trojan infested zombie computers that way, without having to invest time and effort in trying to take over every single computer on Earth.
- Secondly, it takes time to break in somewhere, and the longer it takes the higher the risk (chance) of detection. This doesn't necessarily apply to computer security, or at least not in the same way. While a burglar has a physical appearance that can both be detected visually and have to move in space time to get where it is heading, the electronic signature of a hacker is a lot more elusive. It might be sufficient to impede the burglar. It might not be as effective to slow down a hacker. Does your computer system have multiple layers of security that would require a dedicated hacker to work his way maually past each obstacle? Are your users capable of detecting if something is terribly wrong, rather than just wrong in the way there almost always seems to be something wrong with any computer? If yes, the analogy holds, if not, it doesn't.

Any risk analysis accounts for probabilities. The proper way to do it is as follows (assuming you are familiar with decision theory and second order predicate calculus):

- Let *Adversaries* be the set of entities identified by properties that are relevant to the success of breaking security measures, such as resources, connections, skills. For each *x* in*Adversaries* , let P_{Ax} be the probability that *x* is an adversary to your system. This probability accounts for the risk that someone with the resources of *x* would have the motivation to attack your system. The sum of all P_{Ax} is greater than or equal to zero and might be greater than one (i.e. it might be probable that you have multiple adversaries).
- Let *Roles* be the set of relevant roles an entity might assume while attacking your system. Such roles typically include external hacker, employee, service provider or employee of service provider, burglar, etc. For each *x* in *Adversaries* and each *R* in *Roles* , let $P_{Rx|Ax}$ be the probability that *x* assumes *R* while attacking your system given that *x* is an adversary. This probability accounts for the possibility that the probability that the adversary would assume a certain role might depend on whether he has motivation to attack you (infiltration). More specifically, the probability also

accounts for security measures over which you have limited control, such as your service providers subjecting new employees to screening, how effective your physical security measures are against burglars, etc. The probability that someone with properties *x* will attack your system in the form of *R* equals the product $P_{Ax} P_{Rx|Ax}$.

- Let *Scenarios* be the set of all relevant cryptographic attack scenarios. For each *C* in *Scenario* , *R* in *Roles* and *x* in *Adversaries* , the proposition $C_{Rx|Ax}$ which expresses that *x* will put himself in scenario *C* if he is an adversary and assumes role *R* , will be either true or false, i.e. the probability is either 0 or 1 - if they **can** get in one way, they **will** attempt to get in that way. This doesn't mean that you either will or will not be subject to a *C* kind of attack, but that *x* either will or will not perform a *C* kind of attack given that he is an adversary and manages to assume role *R* .
- For each *C* in *Scenario* , let U_C be the utility of being subject to an attack in *C* , given that such an attack takes place. Positive utility means benefit, negative utility means damage. Each *C* has to be sufficiently well defined for U_C to be independent of which *x* is conducting the attack.
- For each *C* in *Scenario* , let *M(C)* be the security measures that prevent an attack in *C* , and let $U_{C|M(C)}$ be the utility of preventing an attack in *C* , including what you pay the cryptographer, for the cryptographic software and/or hardware, performance degradation etc.
- For each *C* in *Scenario* , *R* in *Roles* and *x* in *Adversaries* , the proposition $C_{Rx\&M(C)|Ax}$ is false if $C_{Rx|Ax}$ is false and true only if *M(C)* does not prevent an attack in *C* with cryptographic security bounds.

If $\sum_{x \in Adversaries} \sum_{R \in Roles} P_{Rx|Ax} (\sum_{C \in Scenarios} U_C C_{Rx|Ax}) \leq \sum_{C \in Scenarios} U_{C|M(C)} + \sum_{x \in Adversaries} \sum_{R \in Roles} P_{Rx|Ax} (\sum_{C \in Scenarios} U_C C_{Rx\&M(C)|Ax})$, then your decision set *M(C)* is adequate. The first term of the right hand expression signifies that you will pay for the security measures regardless of if an attack actually takes place. Both the left hand expression and right hand expression will normally be negative.

This analysis becomes slightly more complex if you are developing off-the-shelf software, rather than in-house software to be deployed at a single location. The probablities, utilities and propositions might be different for different customers. Your damages in case of a security breach are not necessarily equal to those of your customer, $U_{C|M(C)}$ might entail no cost and no damages for some customer but not others, etc. Another question is whether you accept the risk that some customer might be subject to some attacks (that would be too costly to try to prevent generally) or rather recommend them not to use your software and loose a sale.

Key management

The purpose of cryptography is to provide security services that prevent unauthorized entities from gaining access to your data in certain ways. To that end, cryptography relies mainly on two things:

- The security of certain cryptographic primitives or building blocks, such as ciphers, hashes and schemes that combine such primitives in certain ways.
- Correct usage, in particular proper management of the cryptographic keys for any key dependent schemes that are used.

The internal key management of the NexusDB encryption engines are implemented using a chain of key derivation functions and key encryption functions. It is not possible to gain access to the key or keys that are used for the cryptographic primitives that are applied directly to the table data, without direct or indirect access to the table password that unlocks the table. For instance, it is possible to gain access to all of the actual keys, from the table password down to the cipher round keys, if you debug your NexusDB server while the password is entered, either by yourself or by an authorized client user who is trusted with the password and connects to the server while you are debugging it. Deriving the keys without

access to anything but a single, static image of the table cipher text, is however as hard as breaking the primitives using crypto-analysis.

Note 1
Cryptographic security can only be guaranteed to the extent that it is assumed that any entity with access to the process of the NexusDB server also has full access to all passwords and keys. Having access to the process implies full access. If you run the executable on a computer you control, you have access to the process.

The encryption engines protect the physical tables files that are stored on disk. Cryptographic security might be required - and achieved - in the event an adversary gains access to the table files without getting access to the NexusDB server process (or the passwords and keys in some other way). This might realistically happen in at least three scenarios:

1. The device (e.g. server computer) where the database is stored, is stolen. If the server computer is seized while the NexusDB server is still running, the hardware and operating system must at least prevent unauthorized access until the server is shut down and the RAM is cleared. If the adversary starts the server, the NexusDB process will not be able to automatically unlock the tables, or, alternatively, the adversary will neither be able to modify the system that is started automatically when the server is turned on, nor break the disk encryption of the partition where it is stored. Authorized clients will be notified immediately using auxilliary means and will not connect to provide the required passwords.
2. The database is stored with less exclusive privileges than the privileges under which the NexusDB server process is running. Some user who has system privileges that allow him to access the database storage location might not be authorized to access all table passwords.
3. The live backup of the database is stored with less exclusive privileges, e.g. on a network access server or on an online backup service.

Note 2
Use encryption engines when there is a possibility that an adversary might get access to the physical tables files, but you are able to rule out that the same adversary will also get access to the corresponding keys and passwords. Using an encryption engine will protect the confidentiality of the data in the tables, or the integrity of the data, or both.

Cryptographic security ultimately depends on the secrecy of the key. If you use the encryption engines for protecting your data under a certain attack scenario, it is essential that the keys and passwords are managed in such way that they will remain secret even under that particular scenario. This might be diffcult in practice, which makes it important to make sure architecture, deployment and security match, and account for this as early as possible during the development cycle. Some key questions are:

- **Why?** What is the end of the security measures? Does the application author want to make it easier for users to call support than fix issues by themselves? Does the application author want the users to rest assured that their data will remain confidental despite the way the application happens to be deployed?
- **What?** Should the contents remain confidential? Authentic? Should the structure remain confidential or authentic?
- **By whom?** Who is responsible for ensuring that the security requirements are met? The client user or client site administrator? The server administrator? The application author?
- **From whom?** Who is the potential adversary? The client user? The ISP?

- **On whose behalf?** Who has an interest in ensuring security, and what exactly does he stand to loose? If **you** have an interest in certain security, **you** should be given both the opportunity and the responsibility to ensure that security, and hence **you** should be in charge of the key management for that particular security aspect.

Entity/Role	Description
Database storage device	The exact location where the database files are stored, defined by potential access points. In an environment with tight security it might be the logical file directory. In other environments it might be the entire universe.
Database files	The physical table files etc.
Database storage device user	People with access to the database storage device, and hence the database files.
Database content	The logical contents of the database, as seen by an application with full access to the database.
Database structure	The logical structure of the database, e.g. table names, table structures.
Database structure author	The person or persons responsible for designing the database structure.
Database admin user	People with access to the database administration interface, and hence the right to determine the rights of database write users and database read users.
Database write user	People with write access to the database content.
Database read user	People with read access to the database content.
Table password	The key material used for protecting the table contents and table structure.
Database user password	The key material sent to the database server for granting access to tables in a database. If the table passwords are managed by a security monitor, this is the client login to the server, otherwise it is the list of table passwords.
Server device	The location where the server application is stored and executed.
Server device user	People with access to the server device.
Server application	The application that accesses the database and publishes it for client access. There might be multiple tiers of server applications. In such case the back end server access the database, and the front end server listens for client connections. Each server application might be considered to run on a separate device.
Server application author	The person or persons responsible for the server application implementation.
Client device	The location where the client application is stored and executed.
Client device user	People with access to the client device.
Client application	The application that displays the data in human readable form, and possibly allows modifications. In the case of stand alone desktop applications with an embedded server engine, the client application might be identical to the server application. In the case of clients that connect to a remote server application, there might be different client applications for different tasks.
Client application user	People who interact with the client application.

Scenario Usage Examples

Example 1 - Not completely trusted database device user

The server device is a reasonably secure co-located server computer. The end is to protect the confidentiality and authenticity of the data the client users store on the server. The ISP in charge of the co-location service might, hypothetically, at any time shut down the server, insert a boot CD, and get full access to whatever is stored on disk. The scenario might be such that it is feasible to modify data files this way, but infeasible to install or replace executables, without the operating system detecting it on next normal reboot.(* An encryption engine with both confidentiality and integrity should be used, and preferably one which is less vulnerable to historical attacks, such as XCB-AES-256 (Red) or SHACAL-CBC-HMAC-256. The database user passwords are stored client side (or memorized by the client users) and only sent to the server over a transport with both confidentiality and authentication. Note that the answers to the questions "by whom" and "on whose behalf" coincide.

*) For instance, the key used for encrypting the partition with the Windows and Program Files directories is stored in a tamper resistant hardware device, but Inetpub and other data directories are either unencrypted or can be accessed by the ISP for other practical reasons (e.g. backup services). In other words, the server device does not completely coincide with the database device; the former is secured but the latter is not. Be careful not to assume anything without reason. It is not hard to install or modify executables this way *per se* .

Example 2 - Not completely trusted backup device

The server device is a reasonably secure co-located server computer. The entire disk is encrypted, but the ISP in charge of co-location provides a backup service that is only partially trusted. Use a fast encryption engine with confidentiality and integrity, such as AES-CCM-128, AES-CCM-256 or AES-GCM-256, to protect the live backup of the database. For reliability, a second backup is stored elsewhere, in case the ISP managed backup is corrupted and rejected by the encryption engine on restoration.

Example 3 - Stolen or misplaced server side database device

The database device is a reasonably secure co-located server computer. The ISP in charge of co-location is completely trusted for normal operations, but there is a risk the database device will fall into the wrong hands after end-of-life.(* Use any appropriate encryption engine with confidentiality, except AES-GCM-256. The database user password is stored client side.

*) For instance, it is stolen, not completely wiped clean after being replaced as part of normal maintenance, etc.

Example 4 - Not completely trusted database device user

Other threat scenarios are plausible in case of a setup like the one in example 1 with a disjoint server device and database device. For instance, the author of the server application front end might be an external developer without access privileges to the database contents. Use XCB-AES-256 (Red) or SHACAL-CBC-HMAC-256 if there are practical reasons to grant this developer database device user privileges.

Example 5 - Stolen or misplaced client side database device

The advice of example 3 applies also to stand-alone desktop applications where the server device, database device and client device coincide. One significant difference is however that in such a scenario there is no external device where the database user password might be stored safely, so the user should memorize it and enter it manually on each run of the application. DO NOT use the same password you use for anything else (period), unless you are confident the password is not stored anywhere, or is stored securely even under this threat scenario. Usually you are not.

Example 6 - Customers who are unable to let things alone

You are the author of the database and the server application. You want to ensure that the database is only accessed through your application, and, for instance, that the customer does not download EM and modify the data or the structure directly. One reason might be that you do not want them to turn to you for support after they have done so, because they will not admit they have tried to fix things for themselves and you will have to spend hours trying to figure out what has gone wrong.

The right way, from a cryptographic point of view, to achieve this end is for you to host both the server and the database, and thereby physically prevent the users from any unauthorized access to the database files. If this is not an option (e.g. because your customers would have reliability concerns or security concerns with such a solution), and the server device and database device has to be a multi purpose computer system the customer controls exclusively, there is no cryptographically sound solution to the problem. You might however use the AES-CBC-MAC-128 encryption engine with a hard coded table password to obfuscate access to the database. There are no guarantees the customer will not eventually figure out how to extract the password, but it might help reduce the overall problem.