УДК 004.651.52

А. В. Копытов, **А. В. Адаманский**

Новосибирский государственный университет
ул. Пирогова, 2, Новосибирск, 630090, Россия

E-mail: lexus2160@gmail.com; adamansky@gmail.com

РЕАЛИЗАЦИЯ ШИФРОВАНИЯ ДАННЫХ ВО ВСТРАИВАЕМОЙ СУБД EJDB

Речь идет о разработке и внедрении метода шифрования данных во встраиваемой кроссплатформенной документо-ориентированной СУБД EJDB. Основными направлениями исследований является анализ существующих схем шифрования данных в различных СУБД, анализ существующих решений, применимых к архитектуре EJDB, и эффективность их использования.

Ключевые слова: шифрование, ejdb, документно-ориентированные СУБД, AES.

Введение

Работа посвящена реализации шифрования данных в нереляционной СУБД EJDB (http://ejdb.org). EJDB принадлежит к классу встраиваемых систем управления базами данных, которые встраиваются в приложения в виде разделяемой программной библиотеки. Существует определенный класс приложений, где удобно использовать встраиваемые СУБД, это: легковесные приложения, использующие СУБД для доступа к данным, хранимых на сменных носителях (DVD и проч.) и которые сами хранятся на этих носителях; компьютерные игры, хранящие данные игрового процесса и самой игры; большой класс приложений, предназначенных для работы на мобильных устройствах. Достаточно часто требуется соблюдение конфиденциальности хранимых данных от неавторизованного доступа, в частности, это очень актуально для приложений игровой индустрии и мобильных приложений, хранящих персональные данные пользователя. В данном контексте рассматривается задача эффективной реализации шифрования данных во встраиваемой СУБД EJDB.

Алгоритм шифрования должен соответствовать следующим требованиям:

- высокий уровень защиты данных против дешифрования при одновременной экономичности реализации алгоритма при достаточном быстродействии;
- открытый исходный код с возможностью использования в коммерческих продуктах.

Шифрование баз данных – достаточно хорошо изученная задача, и существует множество библиотек, помогающих реализовать шифрование, но более детальное исследование показало, что большинство из них не подходят для реализации поставленной задачи по следующим причинам:

1) часть из решений является платными;
2) некоторые из рассмотренных продуктов разрабатываются под лицензиями, не позволяющими использовать их в коммерческих продуктах. Например, широко используемый программный продукт Cryptlib распространяется под лицензией GPL;
3) несовместимость с архитектурой СУБД EJDB.

Копытов А. В., Адаманский А. В. Реализация шифрования данных во встраиваемой СУБД EJDB // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2014. Т. 12, вып. 2. С. 48–54.

ISSN 1818-7900. Вестник НГУ. Серия: Информационные технологии. 2014. Том 12, выпуск 2

Архитектура EJDB

EJDB (Embedded JSON DataBase engine) [1] является встраиваемой кроссплатформенной документо-ориентированной (иерархической, в англоязычной литературе document-oriented [2]) системой управления базами данных для JSON данных, которая представлет собой разделяемую библиотеку для windows/linux платформ.

Самые популярные СУБД для работы с JSON данными – это mongodb и couchdb, но они работают с использованием протокола TCP/IP. Такой подход очень удобен в реализации серверных приложений, однако не слишком пригоден для встраивания базы в легкие приложения, такие как мобильные приложения и проч., по следующим причинам:

- скорость общения клиента с базой данных через протокол TCP/IP значительно медленнее, чем скорость того же общения, работающего через разделяемую память в разделяемом адресном пространстве запущенного процесса;
- известные препятствия возникают во время развертывания программного продукта в конечной среде. Эти трудности приходят из необходимости одновременного развертывания продукта с СУБД. Таким образом, мы должны установить систему управления базой данных как отдельный сетевой сервис. Эта конфигурация более сложная для поддержки, а также может создавать уязвимости в безопасности программы.

Основой EJDB является общая философия MongoDB и язык запросов этой СУБД. Реализация EJDB технически основывается на tokyocabinet [2] – хранилище данных типа ключ-значение, которое распространяется под лицензией LGPL (Lesser GNU Public License). Система хранения EJDB построена на низкоуровневых структурах данных, которые предоставляет tokyocabinet:

- B+ дерево (tcbdb);
- хеш-таблица, которая хранит записи вида ключ-значение (tchdb);
- табличная база данных (tctdb)

Внутри EJDB JSON документы представлены как BSON (Binary JSON) объекты. Этот формат достаточно компактный и эффективный для хранения и обработки данных.

EJDB хранит набор коллекций, где каждая коллекция содержит набор логически связанных документов JSON. Логически каждая коллекция EJDB – это табличная база данных tokyocabinet (tctdb), которая является хранилищем таблиц, не зависимых от схемы с записанным данными в строки, разделенные на набор определенных пользователем колонок.

Структура документа JSON

TABLE I. JSON DOCUMENT IN TCTDB

Primary key column	BSON document column	Document metadata column
12 byte UUID primary key	BSON document body	Document security attributes, access stats, etc.

Каждая строка таблицы разделена на три части: 24-битный уникальный идентификатор, тело документа в формате BSON и дополнительные метаданные документа.

Если документы не содержат проиндексированные поля в процессе выполнения запроса, вся коллекция документов будет просканирована, и BSON-документы, чьи поля подошли к запросу, будут выбраны. Выбор документа по первичному ключу сводится к выборке записей данных с диска при помощи хэш-таблицы. Эта операция может быть выполнена достаточно быстро. EJDB-запросы определены как набор CRUD-операций с документами, хранящимися в коллекциях, т. е. создание, чтение, обновление и удаление. Правила CRUD представлены в виде набора BSON-документов, состав которых похож на запросы MongoDB. CRUD-операции чтения определяют набор ограничений на поля, применяемых к коллекции документов. Так как структура JSON-документов может быть иерархичной, то fieldpath ис-

[1] Knowledge Base of Relational and NoSQL Database Management Systems. URL: http://db-engines.com/en/ranking_categories

[2] Tokyo Cabinet: a Modern Implementation of DBM. URL: http://fallabs.com/tokyocabinet/

пользуется для идентификации документа. Например, условие запроса на выбор книг с определенным издателем может быть таким: {"publisher.name" : "some publisher"}.

EJDB стремится быть совместимой с MongoDB в плане запросов, так как это обеспечит возможность легкой миграции приложений из/в MongoDB. На данный момент около 70 % запросов mongodb реализованы в EJDB, планируется достигнуть совместимости по запросам в 90 %.

Кроме того, EJDB расширяет возможности запросов MongoDB и предоставляет следующие возможности извлечения данных:

- оптимизированное сравнение строк, не зависящее от регистра;
- быстрое сравнение строк, например, запрос *{"words" : {"$strand" : ["one", "two"]}}* находит докумнты, где строковое поле *words* содержит слова *«one» и «two»* в наборе из слов, разделенных пробелами.
- оптимизированный строковый префикс matching with *"$begin"*;
- соединение набора документов. Вхождения документов в различные коллекции могут быть объединены с помощью первичного ключа, как результат одного запроса. Если разработчик захочет получить объединение объектов из связанных коллекций, ему придется исполнить как минимум $N + 1$ запрос, где N – это количество элементов основной коллекции. Для того чтобы ликвидировать эти тривиальные раунды, EJDB предлагает способ указать объединение как часть исполнения одного запроса в следующей формой: *"{$do : {<fpath> : {$join : <collection name>}}}",* где *<fpath>* – это путь к полю документа, содержащего идентификаторы объектов, которые будут объединены с другой коллекцией.

Шифрование

Поскольку асимметричное шифрование затратно по ресурсам и времени, то на практике обычно для шифрования данных используется симметричный, блочный алгоритм. Рассмотрим различные режимы функционирования блочных шифров.

ECB (*Electronic Codebook*). В данном режиме шифрования последовательность открытого текста разбивается на блоки одинакового размера и каждый из блоков шифруется независимо друг от друга одним и тем же ключом. Шифрование в режиме ECB выполняется быстро, но в связи с тем, что для каждого блока используется один и тот же ключ, существует один важный недостаток: если получится, что в исходном тексте найдутся два одинаковых блока, то результатом шифрования тоже будут два одинаковых шифрованных блока. В данном случае злоумышленник сможет сделать некий вывод о содержании исходных данных, что может существенно облегчить криптоанализ.

CBC (*Cipher Block Chaining*). Для исправления этого недостатка разработан режим сцепления блоков шифротекста, который предусматривает, что данные так же разбиваются на блоки одинакового размера, но шифрование происходит иным образом. Перед тем как зашифровать очередной блок к исходному тексту, добавляется путем суммирования по модулю 2 шифрованный блок, полученный на предыдущем шаге. Таким образом устраняется основной недостаток режима ECB. Чтобы избежать этого недостатка в первых блоках, к ним добавляется случайный вектор, который называется вектором инициализации.

Выбор алгоритма шифрования

На основании вышеизложенного подходящим для нашей задачи вариантом является симметричный блоковый шифр. В контексте базы данных EJDB нам нужен алгоритм с максимальной производительностью при достаточной криптостойкости.

За основу были взяты исследования, описанные в [2] и [3]. В результате проанализированной информации был выбран алгоритм Rijndael, который на данный момент и является американским стандартом шифрования AES (Advanced Encryption Standard).

Схема шифрования

На основании изученной информации был сформирован следующий способ шифрования. Во-первых, чтобы не осложнять себе задачу хранения секретного ключа, ключ будет генери-

роваться на основе информации, которая хранится в голове у пользователя – его пароля. Но так как мы не можем использовать в качестве ключа сам пароль, потому что ключ должен состоять из абсолютно случайной последовательности и его нельзя было бы взломать статистически (как мы знаем большинство паролей, создаваемых пользователями, не удовлетворяет этому свойству), то ключ будет генерироваться по специальной схеме (рис. 1).

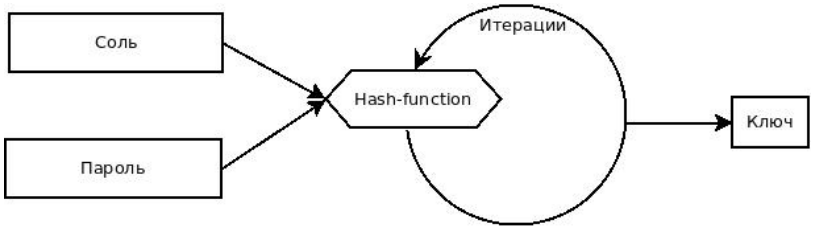

Рис. 1. Схема генерации криптографического ключа

Рекомендации по генерации ключа, на основе пароля были взяты из документа NIST [3]. Для генерации ключа нам понадобится криптографическая соль. Соль представляет собой случайносгенерированную последовательность байтов и хранится в открытом виде. Пароль объединяется с солью {password@salt} и хэшируется *N* количество раз. Количество итераций может быть различным, как 10, так и 10 000 000. Чем больше количество итераций, тем более защищенной будет наша система, но в то же время тем дольше будет она генерировать ключ. В данном контексте на основе эксперимента была выбрана величина в 1024 итерации. Такая конфигурация ключа позволяет защититься от радужных таблиц и других способов подбора пароля. Основная идея состоит в том, чтобы замедлить атаки по словарю или атаки полным перебором за счет увеличения времени, которое понадобится для тестирования каждого пароля.

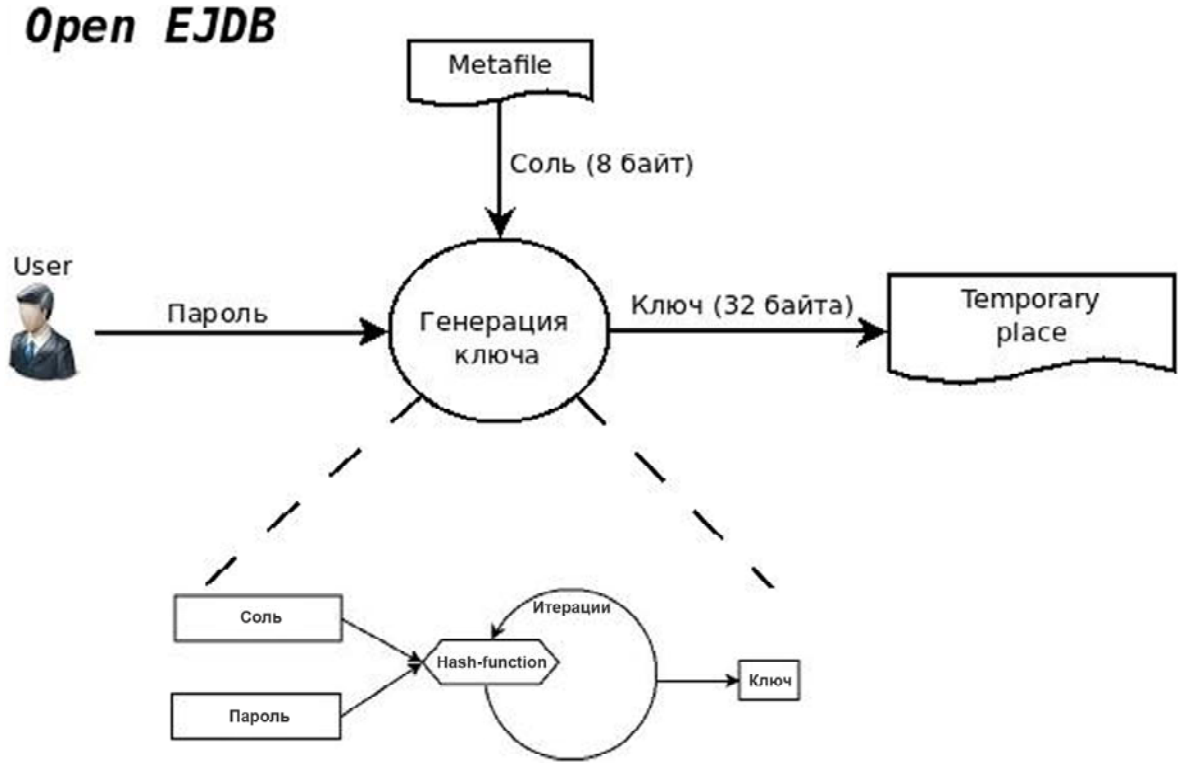

Рис. 2. Схема открытия EJDB

Открытие EJDB. При открытии базы данных происходят следующие действия (рис. 2).

1. Пользователь вводит пароль.

[3] Report on the Development of the Advanced Encryption Standard (AES), National Institute of Standards and Technology, October 2, 2000.

2. Если пользователь открывает базу впервые, генерируется криптографическая соль, иначе соль загружается из метаданных.

3. Генерируется криптографический ключ на основе пароля и соли по схеме описанной выше.

Сохранение. При сохранении очередной записи в базу данных происходит следующее (рис. 3).

1. Генерируется инициализирующий вектор (ИВ) хэшированием уникального идентификатора записи (ИВ = Hash(UID)).

2. С помощью ключа и инициализирующего вектора исходные данные шифруются в режиме CBC (cipher-block chaining), блоками по N байт ($N = 16$).

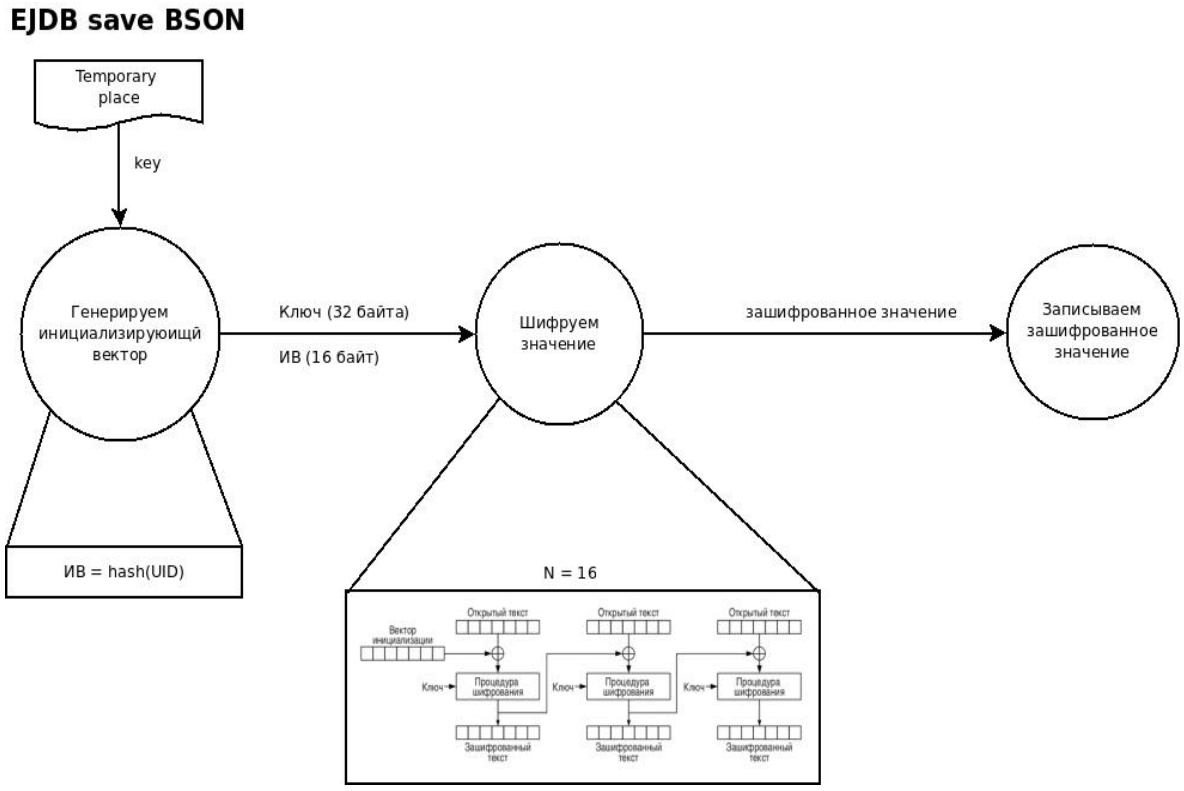

Рис. 3. Сохранение записи в EJDB

Таким образом, шифрование происходит прозрачно от пользователя, т. е. он, так же как и в незашифрованной базе данных, записывает, считывает данные, не задумываясь о том, включено шифрование или нет. Все, что нужно сделать, – это указать настройки при создании базы.

Загрузка. Дешифрование происходит аналогичным образом, с помощью того же инициализирующего вектора и ключа шифрования, сгенерированного на основе пароля пользователя. Отметим, что данный вид шифрования базы данных предназначается для защиты базы в то время, когда она не используется (украденный персональный компьютер и т. п.) (рис. 4). Шифрование не спасет в случаях, когда у атакующего есть доступ к файлам в то время, когда база данных используется.

Также заметим, что если пользователь в качестве пароля выберет комбинации наподобие «12345» или «qwerty», то никакие средства защиты не помогут, ни соль, ни хэш, ни какие-либо еще средства шифрования. Этот момент мы оставляем на усмотрение пользователя. Такой пароль с помощью атаки по словарю взламывается за считанные секунды.

Кроме того, при таком типе шифрования, когда ключ генерируется на основе пароля, если пользователь забудет пароль, то у него уже не будет никакой возможности восстановить зашифрованные данные.

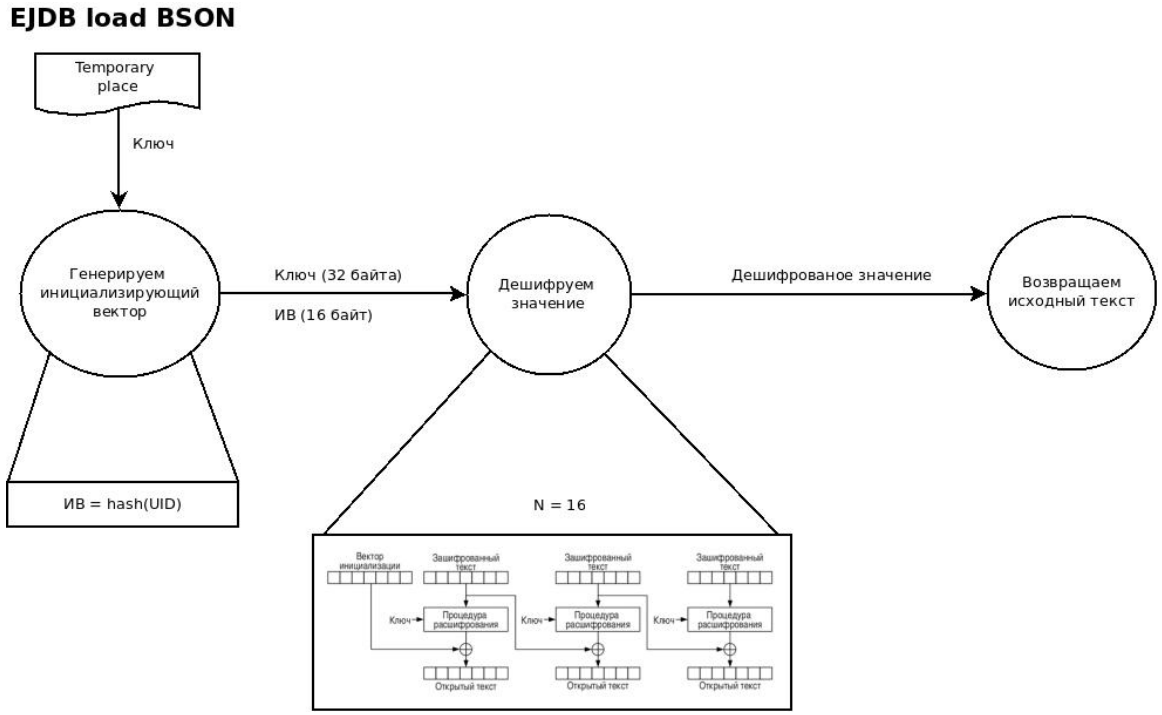

Рис. 4. Загрузка записи в EJDB

Итоги работы

Разработанная схема была реализована с использованием алгоритма шифрования AES с размером ключа 128 и алгоритмом хеширования SHA-256.

Включение прозрачного шифрования ожидаемо дало ухудшение по времени. На рис. 5 приведены сравнительные графики.

Без шифрования	AES	
715	**1345**	**1.8811188811**
5786	**44664**	**7.7193225026**
355	**781**	**2.2**
2	**2**	**1**

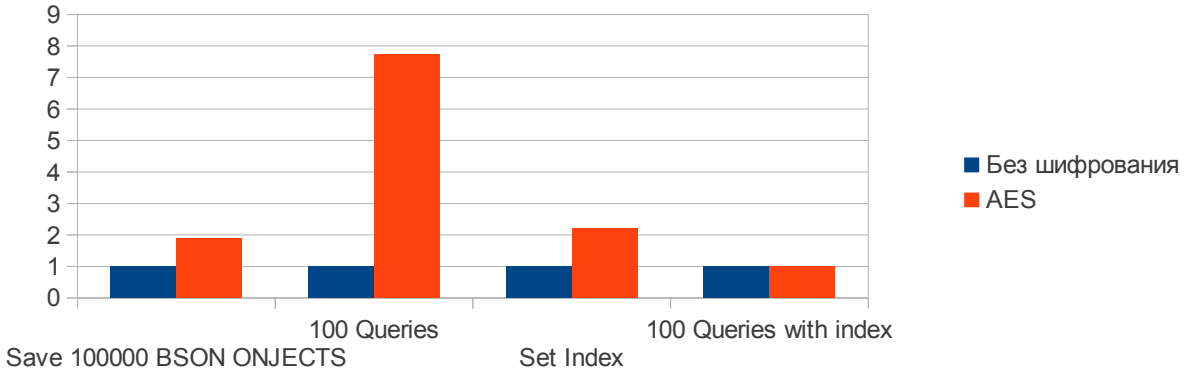

Рис. 5. Результаты

Как видно, в самом плохом случае наблюдается ухудшение в 7 раз, что в целом приемлемо в рамках поставленной задачи.

Таким образом, удалось реализовать схему шифрования, удовлетворяющую всем указанным в начале статьи критериям.

Список литературы

1. *Adamanskiy A., Denisov A.* EJDB – Embedded JSON database engine // Fourth World Congress on Software Engineering (WCSE). 2013. P. 161–164.

2. Report on the Development of the Advanced Encryption Standard (AES). National Institute of Standards and Technology, 2000.

3. *Винокуров А., Применк Э.* Сравнение стандарта шифрования РФ и нового стандарта шифрования США. URL: http://www.enlight.ru/crypto/articles/vinokurov/gosaes_i.htm

Материал поступил в редколлегию 24.04.2014

A. V. Kopytov, A. V. Adamanskiy

IMPLEMENTING DATA ENCRYPTION IN EMBEDDED DATABASE EJDB

The article focuses on the development and implementation of methods of data encryption in embedded cross-platform document-oriented database EJDB. The main areas of research was to analyze the existing data encryption schemes in different databases, and to analyze of existing solutions that are appropriate to the EJDB architecture and efficiency of their use.

Keywords: encryption, ejdb, document-oriented database, AES.

References

1. Adamanskiy A., Denisov A. EJDB – Embedded JSON database engine. *Fourth World Congress on Software Engineering* (*WCSE*), 2013, p. 161–164.

2. Report on the Development of the Advanced Encryption Standard (AES). National Institute of Standards and Technology, 2000.

3. *Vinokurov A., Primenko E.* Comparison encryption standard of Russia and new encryption standard of USA. URL: http://www.enlight.ru/crypto/articles/vinokurov/gosaes_i.htm