Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Внедрение шифрования в систему хранения данных высокой производительности

Курсовая работа студента 345 группы

Овчинникова Антона Андреевича

Научный руководитель ……. Ершов П. А., AvroRAID,

инженер по тестированию

Санкт-Петербург

2011

Оглавление

Введение

В наши дни информация становится все более и более ценным товаром, и уделение должного внимания сохранности этого товара часто является критической необходимостью.

Шифрование каналов связи позволяет на порядок уменьшить возможность нежелательного доступа к данным, но не менее важным является и обеспечение безопасности информации «в состоянии покоя» (data-at-rest).

Каналы передачи данных могут обеспечивать отличный уровень защиты, но какой в этом смысл, если на жестком диске, где информация будет храниться после передачи по упомянутому каналу, будут вообще отсутствовать какие-либо средства для предотвращения нелегального доступа?

Целями работы являются исследование особенностей шифрования накопителей в системах хранения данных (СХД), а также поиск оптимального варианта встраивания механизма шифрования в конкретную систему на примере СХД Avrora.

1. Описание требований

Системы хранения данных являются специализированными средствами для достижения целей хранения и передачи информации. Они нуждаются в эффективных способах реализации и внедрения шифрования.

Но одно дело — изначально создавать систему хранения с поддержкой шифрования, и другое — правильно организовать интеграцию этого механизма с уже существующей и успешно функционирующей СХД. Попытка серьезного изменения архитектуры системы может повлечь существенную трату времени и материальных ресурсов.

Каким еще свойствам должна удовлетворять Система Хранения с шифрованием?

***1. Конфиденциальность*:** не допускается получение хранимой информации лицом, не являющимся ее владельцем. К основным угрозам в данном контексте относятся кража (или изъятие) всей СХД, длительное наблюдение за «сырыми» данными, прослушивание канала от инициатора к СХД, а также попытки «внутреннего» проникновения (то есть попытка получить доступ с инициатора). В конечном итоге, все приведенные варианты не должны ставить под угрозу хранимую информацию.

2. Производительность: считывание и запись информации с/на диск должны выполняться быстро, причем вне зависимости от местоположения данных, то есть достижение произвольности доступа не должно отразиться на производительности.

3. Доступность: пользователь должен иметь беспрепятственный доступ к данным, к которым он имеет право обращаться.

4. Целостность данных: состояние данных может быть изменено только субъектами, имеющими на это право, а значит, данные не должны никаким образом меняться в процессе передачи, хранения и извлечения.

5. Рациональность использования: способ шифрования не должен использовать слишком много места на дисках**.** Это означает, что размер зашифрованных данных вместе с такой информацией, как ключи и необходимые для алгоритмов шифрования другие данные не должны значительно превышать размер шифруемого открытого текста.

6. Прозрачность для пользователя: система с шифрованием не должна требовать более серьезного вмешательства со стороны для функционирования по сравнению с системой без шифрования (лучше всего – просто ограничиться запросом для ввода аутентификационных данных).

2. Выбор режима шифрования

Алгоритмы шифрования обычно обрабатывают блоки данных фиксированной длины (например, алгоритм AES работает над блоками размером 16 байт). Если же нужно зашифровать данные произвольного размера, необходимо использование режима шифрования – метода, согласно которому алгоритм шифрования будет обрабатывать эти самые данные.

Самые известные режимы — ECB (Electronic codebook), CBC (Cipher-block chaining), CFB (Cipher feedback), OFB (Output feedback) – являются непригодными для применения в Системах Хранения, так как имеют серьезные уязвимости в контексте нашей специфичной задачи. Например, режим ECB имеет существенный недостаток: одинаковые блоки открытого текста шифруются в одинаковые шифротексты (при использовании одного ключа, разумеется). Другими словами, режим электронной книги не в состоянии скрыть повторяющиеся куски открытого текста, а это уже уязвимость: зная даже приблизительную структуру зашифрованных данных (например, выделяя стереотипные заголовки файлов или email-сообщений) криптоаналитик или атакующий может получить много информации о содержании открытого текста. А на режим CBC существуют несколько серьезных атак: watermark-атаки, copy&paste и другие.

Из-за существенных уязвимостей «классических» режимов шифрования стало необходимым разработать специализированные режимы, предназначенные для использования в накопителях. Одним из первых таких режимов стал режим LRW (Liskov, Rivest, Wagner)[1]. Его создатели представили целый класс «настраиваемых блочных шифров», которые используют специальный битовый вектор (размер которого обычно совпадает с размером блока для шифрования) — твик (tweak). Он служит скорее не для усиления криптостойкости, а для рандомизации выходного шифротекста при одинаковых открытых текстах. Успешной реализацией этой концепции стал режим XEX (XOR-Encrypt-XOR), предложенный в 2006 году, но сейчас более известной является его модификация: XTS (**X**EX-based **T**weaked encryption mode with ciphertext **S**tealing). Рассмотрим его более подробно как наиболее перспективный режим для применения в шифровании накопителей.

Пусть у нас есть блок данных C размером 16 байт (предполагаем, что используем алгоритм шифрования с таким размером блока), который является j-м блоком внутри более крупной структуры данных (назовем ее сегментом). Сегмент, в свою очередь, имеет свой порядковый номер внутри области действия ключа (keyscope), то есть некоторого объема данных, шифруемых одним ключом.

Тогда:

C =XTS-BlockEnc (Key,P,i,j) := Enc-Key1(M $\oplus$ T) $\oplus$ T;

где T = Enc-Key2 (i) $\otimes$ α^j

Здесь:

C – шифротекст (128 бит)

T – твик (128 бит)

M – открытый текст (128 бит)

j – логический индекс блока внутри сегмента данных (128 бит)

Key1, Key2 – по 128, либо по 256 бит каждый.

i – твик-вектор, размер которого совпадает с размером блока (128 бит), а по сути – это целое число, которое присваивается сегментам данных последовательно, начиная с некоторого произвольного неотрицательного целого числа. В простейшем случае, это номер сегмента данных, начиная с 0.

$\otimes$ - умножение в поле GF (2^{128}) по модулю $x^{128}+x^7+x^2+x+1$

Вычисление T можно не проводить каждый раз заново, нужно только домножать текущее значение T на α при переходе к следующему блоку.

Этот режим уже несколько лет успешно применяется для шифрования накопителей в таких программных продуктах, как Truecrypt, BestCrypt, dm-crypt др.

3. СХД Avrora

В дальнейшем будем опираться на более конкретный пример: СХД Avrora. Это российская СХД, ориентированная на высокую производительность и низкую стоимость за счет использования стандартных компонентов. Avrora позволяет создавать RAID уровней 0, 10 и 6, причем эффективная реализация последнего является одним из главнейших достоинств этой СХД.

Возникла задача внедрения шифрования в Avrora, и так как одним из важнейших преимуществ данной СХД является высокая производительность, главным моментом было не допустить деградации производительности.

В качестве алгоритма шифрования был выбран алгоритм AES (Rijndael), поскольку он является современным и хорошо проанализированным алгоритмом, а также из-за отличной аппаратной поддержки со стороны процессоров Intel: новый набор инструкций AES-NI показывает в среднем четырехкратный выигрыш в производительности по сравнению с программной реализацией. В скором времени поддержку этих инструкций будут осуществлять и процессоры AMD.

Режимом шифрования было решено сделать уже упомянутый XTS, причем областью действия ключа (keyscope) делается целый LUN (Logical Unit Number), что в какой-то степени аналогично шифрованию одного раздела в программах вроде Truecrypt, BestCrypt. Связано это с тем, что обычно пользователям СХД нужно обращаться к конкретному LUN. В каждом LUN – собственная адресация LBA (logical block address), что позволяет независимо вычислять номера сегментов (некоторое количество последовательных блоков, которые объединяются для проведения операций в режиме XTS) и блоков внутри сегментов.

4. Управление ключами

Данная тема сама по себе является довольно обширной, потому были рассмотрены только базовые концепции. В целом, подход к управлению ключами зависит от конкретной задачи и сферы применения СХД.

Как уже было упомянуто, каждый LUN шифруется отдельным ключом, то есть для доступа к каждому из шифрованных LUN необходимо пройти авторизацию.

Есть несколько проблем:

a) Низкая энтропия данных доступа. Речь, в первую очередь, идет о «слабых» пользовательских паролях. Этой проблемы можно избежать, используя политики паролей или биометрические данные.

b) Даже с учетом пункта 1, трудности возникают, когда идентификатор доступа нужно сменить (пароль) или обновить (например, отпечаток пальца). Ведь придется сначала расшифровывать весь диск с помощью старого идентификатора и зашифровывать с помощью нового.

c) У разных LUN могут быть разные алгоритмы шифрования и разные длины ключей.

Для предотвращения этих трудностей используются метазаголовки и двухступенчатая схема аутентификации. Метазаголовки – это метаданные специального типа, существующие для каждой области действия ключа (для каждого LUN в нашем случае), в котором будет содержаться информация об используемом алгоритме, режиме шифрования, а также ключ для шифрования всего остального раздела (мастер-ключ). А уже сам этот заголовок шифруется с помощью пользовательского ключа, над которым сначала проводят специфичные операции для увеличения энтропии.

До недавнего времени не было единого стандарта, который бы описывал эти метаданные, и каждая программа использовала свой формат заголовка.

Но в 2009 году появилась первая версия спецификации LUKS (Linux Unified Key Setup), которая может вскоре стать единым стандартом для шифрования накопителей. LUKS предоставляет документированную, платформо-независимую спецификацию, описывающую формат мета-заголовка, а также процедуры установки, смены и удаления ключей. Также LUKS позволяет создавать до 8 паролей для одного шифруемого раздела. Если этого окажется мало, данная спецификация допускает масштабируемость.

По произведенным оценкам, размер метаданных при использовании метазаголовка формата LUKS не превосходит 10 Мегабайт (большую часть места занимает информация, порожденная AF-Splitter – особым методом, значительно уменьшающим вероятность

восстановления пароля после его удаления). При максимальном количестве LUN, равном 512, получаем около 5 Гбайт метаданных, что является небольшим объемом по сравнению с тем, на который ориентирована СХД.

Сразу же встает вопрос: где хранить метаданные для каждого LUN? Можно в начале каждого LUN держать его мета-заголовок, таким образом, для «открытия» этого логического раздела не потребуется больше никуда обращаться. Но у этого подхода есть недостаток: имея полный доступ к LUN, нужно принимать меры для предотвращения изменения заголовка. Это возможно реализовать, но обратной стороной будет то самое нежелательное изменение архитектуры. Гораздо более перспективным выглядит метод, при котором все метаданные хранятся на отдельном логическом разделе. При таком подходе достигается централизованность хранения ключей, что позволяет быстро их изменять и удалять (это также можно использовать как дополнительную предосторожность при нежелательном вскрытии системы: без метаданных у атакующих будет намного больше проблем в плане определения алгоритма шифрования, длины ключа, используемой хеш-функции и т.п.).

Также к проблеме управления ключами можно отнести протокол аутентификации, то есть как поэтапно будет происходить общение пользователя с системой.

При отсутствии шифрования пользователь на инициаторе сразу видит СХД как подключенный LUN (чаще всего, как отдельный жесткий диск), с которым сразу же можно обмениваться данными. В нашем случае ситуация другая: необходима специальная программа, которая будет выполняться на клиентской машине и взаимодействовать с Системой Хранения. В простейшем виде это будет выглядеть следующим образом:

1) Пользователь запускает на инициаторе специальную утилиту
2) Выбирает логический раздел, к которому хочет получить доступ
3) Вводит логин и пароль (биометрические данные, вставляет смарт-карту), которые передаются на таргет.
4) Специальный модуль производит обработку полученных данных, которые сравниваются с метаданными из соответствующего заголовка LUN (при использовании формата LUKS эти операции четко описаны).
5) В зависимости от результата предыдущего шага, пользователю дается или запрещается доступ.

У этой схемы (в приведенном виде) есть уязвимое место: на третьем шаге идентификационные данные передаются в открытом виде. Если есть большая вероятность прослушивания канала от инициатора к таргету, можно использовать схемы с использованием асимметричного шифрования. Причем не обязательно вводить громоздкую инфраструктуру обмена открытыми ключами, достаточно обеспечить парой

открытой/закрытый ключ только сам таргет. Таким образом, данные от инициатора(пароли) передаются по каналу надежно зашифрованными.

Другой, также приемлемый вариант, состоит в том, чтобы использовать внешний сервер ключей для хранения открытого ключа. Это будет дополнительной мерой предосторожности от подмены ключа и атак типа «человек посередине» (man-in-the-middle).

Таким образом, основным моментом в вопросе аутентификации является организация некоего протокола, позволяющего организовать эффективное взаимодействие между пользователем и СХД. Он может быть организован по типу «запрос-ответ». Преимуществом введения такого протокола также будет расширяемость: можно будет дополнить его опциями смены пароля, создания нескольких паролей для одного логического раздела и т.д.

5. Уровень встраивания

Вкратце рассмотрим архитектуру СХД Аврора. Основной частью является модуль hyperraid_mod, который осуществляет поддержку RAID6: вычисление 2х синдромов (разных контрольных сумм, необходимых для восстановления информации в случае выхода из строя до 2х дисков). «Над» ним находится SCSI-драйвер scst_nr, который отвечает за отображение LUN на RAID и создает абстракцию для работы по разным аппаратным интерфейсам: FibreChannel, Infiniband и др. Выше находятся драйвера-связки с определенными физическими интерфейсами.

Можно выделить два принципиальных подхода при встраивании механизма шифрования:

1. Встраивание в RAID-Модуль (в нашем случае – hyperraid_mod), то есть шифрование будет происходить прямо перед записью или сразу после чтения. Важно учесть: шифрование должно происходить ДО вычисления синдромов, так как иначе возникает некоторая зависимость между зашифрованными данными, что позволяет применить разные техники криптоанализа. К тому же, достигается выигрыш в производительности: нужно меньше шифровать. А главным достоинством этого подхода является именно выигрыш в скорости, пока, правда, теоретический. Имеет место локальность обращения к памяти, Велика вероятность попадания в кэш нужных для последующих вычислений данных.
2. SCSI-модуль. Как только приходит SCSI-команда (Чтение, Запись), происходит извлечение данных (из команды при записи, с диска при чтении), которые затем шифруются/расшифровываются. Этот подход проще реализовать, так как нет необходимости писать реализацию под каждый RAID-уровень. Но есть и недостатки: по сути, происходит несколько «пробегов» по данным, первый раз – при шифровании, второй – при вычислении синдромов.

6. Результаты тестирования

После проведения исследовательской части был реализован метод шифрования с использованием AES с длиной ключа 256 бит в режиме XTS с применением инструкций AES-NI и Intel intrinsics. После этого было проведено тестирование, в ходе которого проверялись зависимость потери производительности расчетов RAID-6 от размеров блоков страйпов, от количества дисков, а также от места встраивания при наличии шифрования AES-XTS (ключ 256 бит). Испытания проводились на тестовой машине с процессором Intel Core i5 520UM с поддержкой набора инструкций AES-NI, операционная система Red Hat Enterprise Linux 6.0. После этого был проведен анализ полученных результатов: в частности, были построены графики полученных абсолютных и относительных значений. Стоит подчеркнуть, что условия тестирования постарались приблизить к реальным (в работающей СХД тестирование пока не проводилось). Потому могут наблюдаться некоторые отклонения от тех значений, которые были бы зафиксированы на реальной системе.

В первую очередь тестировалась производительность при расчете синдромов P и Q для RAID-6, так как новые, поступившие в СХД данные обязаны пройти через эту процедуру. Было реализовано два подхода: шифрование всего страйпа перед вычислением синдромов (в какой-то степени моделирование подхода шифрования на SCSI-уровне) и шифрование блока прямо перед вычислением синдромов (второй подход, RAID-уровень). Результаты оказались интересными: ожидалось, что второй вариант покажет более высокую скорость (по той же причине, почему более эффективным считается шифрование на RAID-уровне, а именно из-за локальности обращения), но на деле выигрыш получался очень незначительным. Произошло это, скорее всего, из-за оптимизаций со стороны кэша. Стоит также учитывать, что тестирование проводилось в user-mode, в то время как реальные драйвера будут работать в kernel-mode.

Из-за незначительности различий, будут использоваться результаты для первого подхода.

Были протестированы: размеры блоков страйпов – с 4 Кб по 128 Кб, количество дисков – от 4 до 120, замеры проводились в тиках (tick) процессора.

Тесты продемонстрировали следующее:

На падение скорости практически не влияет размер блоков страйпов, что хорошо видно из графиков 1 и 2, где падение показано «в разах».

Можно видеть, что в зависимости от количества дисков падение производительности может быть различным: основной тенденцией является уменьшение коэффициента падения при увеличении количества дисков.

Табл. 1 Генерация синдромов P и Q с шифрованием и без

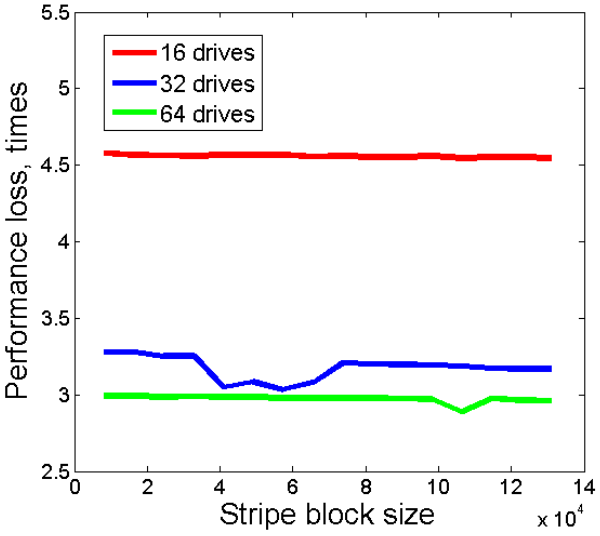

Рис. 1 Потеря производительности, страйпы-диски

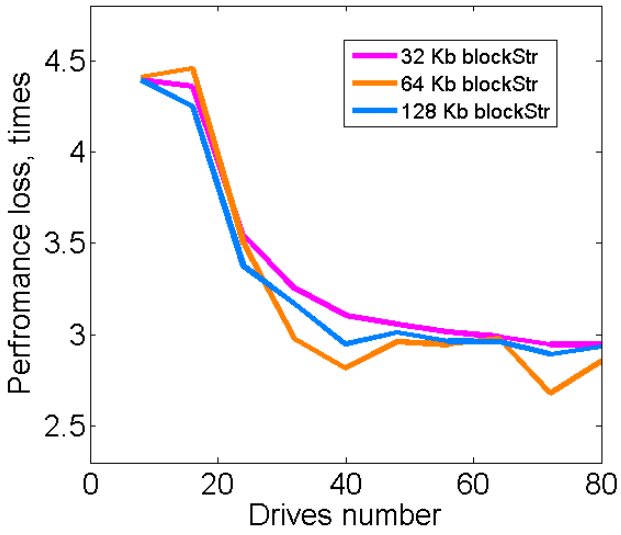

Рис. 2 Потеря производительности, диски-страйпы

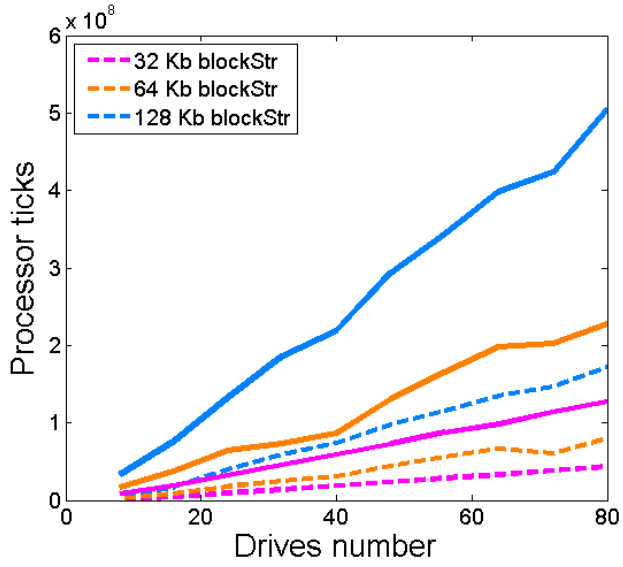

Рис. 3 Абсолютные значения, диски-страйпы

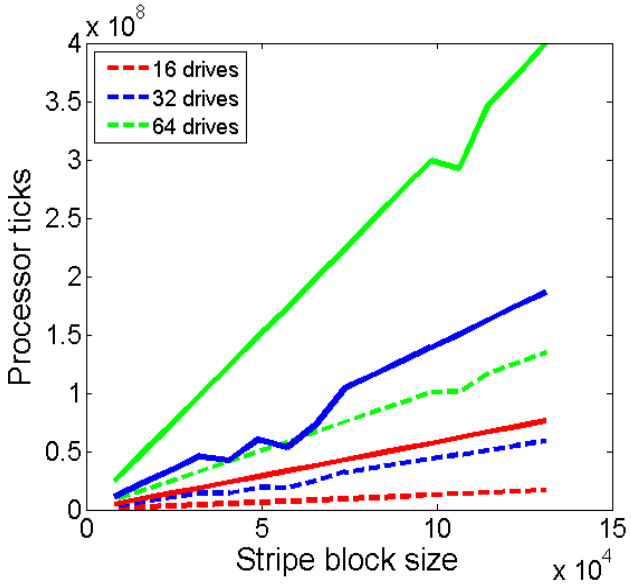

Рис. 4 Абсолютные значения, страйпы-диски

Когда дисков немного (до 20), вычисление синдромов происходит до 4.5 раз медленнее. При увеличении количества дисков это число уменьшается примерно до 2.5. Связано это с тем, что два сравниваемых процесса с определенными допущениями имеют вид линейных функций вида (ax + b) и (cx+d), у которых (a * d) != (b * c). То есть графики этих двух функций не пересекаются в нуле, что видно на рисунке 3: там показаны абсолютные значения вычисления синдромов с шифрованием и без (пунктирные линии) при

фиксированных размерах блоков страйпов. В итоге мы получаем приближенную гиперболу на графике 2.

Табл. 2 Сравнение чтения и чтения с расшифровыванием

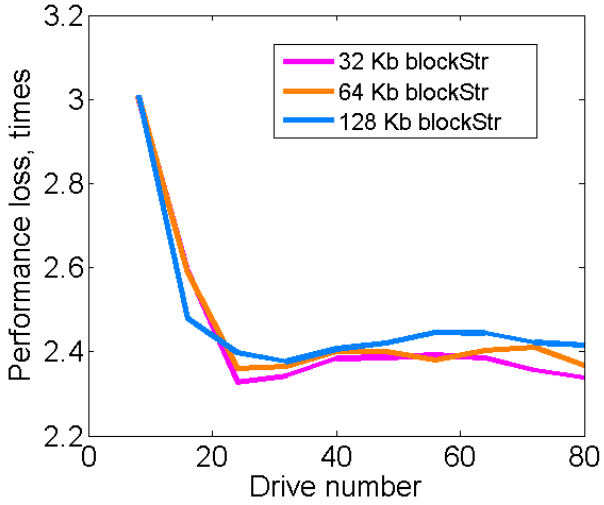

Рис. 5 Потеря производительности, диски-страйпы

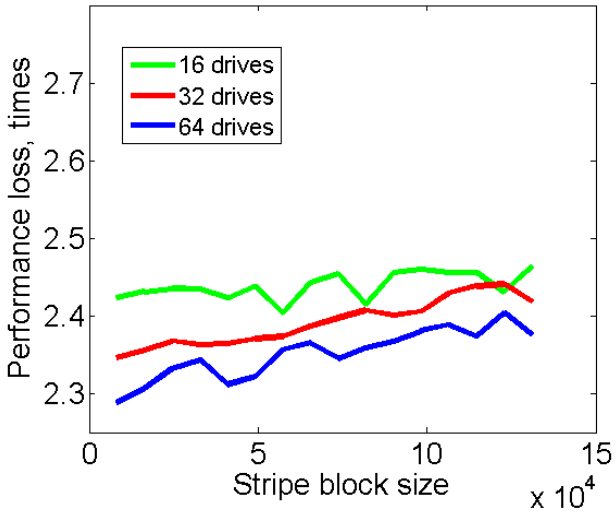

Рис. 6 Потеря производительности, страйпы-диски

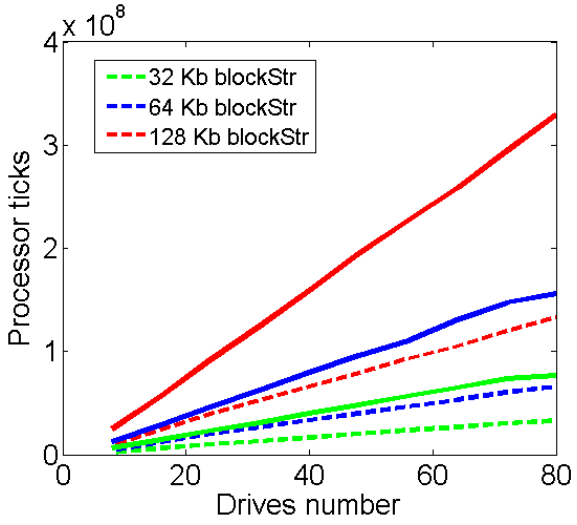

Рис. 7 Абсолютные значения, диски-страйпы

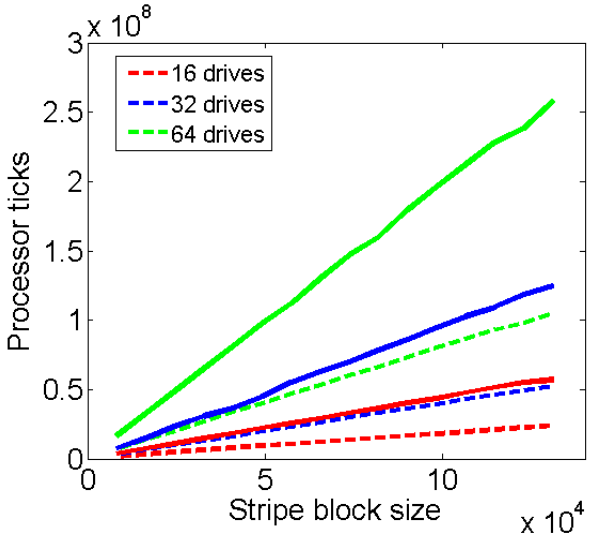

Рис. 8 Абсолютные значения, страйпы-диски

Также было протестировано расшифровывание, которое сравнивалось с обычным чтением. Результаты получились похожие, что видно из графиков 5 и 6. Наблюдается снижение (незначительное, правда) коэффициента падения производительности при увеличении количества дисков. На графиках 7, 8 приведены абсолютные значения.

Таким образом, тестирование показало, что использованное шифрование не должно существенно сказаться на производительности в реальной системе: вычисление синдромов является довольно незначительной в плане влияния на быстродействие операцией, самым

узким местом в СХД обычно являются сами накопители, и многое зависит от алгоритмов кэширования, реализованных там. Поэтому увеличение времени при вычислении синдромов даже на порядок не должно показать серьезное падение производительности. Больше на данный момент сказать трудно, решающим станет финальное тестирование на реальной системе, так как есть много факторов, которые трудно учитывать на стадии моделирования: опять же поведение кэша, скорость чтения/записи на диск и т.д.

Заключение и выводы

Было проведено исследование применения шифрования в СХД, определены требования, выделены особенности и отличие от шифрования каналов данных.

Проанализированы возможные варианты внедрения шифрования в СХД Avrora.

Тестирование показало, что операции (рас)шифрования при эффективной реализации не окажут существенного влияния на производительность всей системы. Также тестирование продемонстрировало незначительное отклонение производительности при двух методах внедрения. Если эти ожидания действительно оправдаются, то встраивание можно будет проводить там, где это проще сделать – на уровне SCSI.

Следующим этапом работы будет внедрение и тестирование на реальной рабочей системе.

Используемые источники

[1] Bruce Schneier, “Applied Cryptography” (1996)

[2] IEEE P1619/D16 “Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices”, 2007

[3] Intel “Advanced Encryption Standard (AES) Instructions Set” White Paper (2010)

[4] Linux Unified Key Setup (LUKS): http://code.google.com/p/cryptsetup/

[5] Liskov, Rivest, Wagner, “Tweakable Block Ciphers” (Advances in Cryptography, 2002)

[6] Компания Avroraid: http://avroraid.com/