ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ

АРХИТЕКТУРА ПАРАЛЛЕЛЬНОГО КРИПТОЯДРА ДЛЯ СИСТЕМ ЗАЩИТЫ ИНФОРМАЦИИ ПРИ ЕЕ ХРАНЕНИИ

А.В. Раевский

Кафедра кибернетики и мехатроники
Российский университет дружбы народов
ул. Миклухо-Маклая, 6, Москва, Россия, 117198

В статье рассматриваются особенности проектирования параллельного криптоядра для системы защиты информации в процессе ее хранения. Приводятся варианты решения таких вопросов, как оптимизация алгоритма шифрования, определение уровня распараллеливания шифрования, выбор криптографического алгоритма и размера блока шифрования. Учитываются особенности использования кэш-памяти, создания и синхронизации потоков. Предлагаются подходы для определения количества рабочих потоков для шифрования.

Оптимизация алгоритма шифрования

Как правило, наиболее очевидный первый шаг в оптимизации любой задачи — это реализация оптимизируемых алгоритмов таким образом, чтобы максимально использовать регистры процессора и кэш-память вместо использования основной памяти. Данный метод может быть довольно эффективным, поскольку операции обращения к основной памяти подразумевают отправление запроса по шине и ожидание ответа [4]. Такие операции могут выполняться в разы дольше, чем обращения к регистрам, которые находятся непосредственно на кристалле ЦП. Поэтому часто имеет смысл либо реализовывать алгоритм шифрования изначально, либо провести его рефакторинг таким образом, чтобы максимально использовать регистры ЦП, по крайней мере, в качестве локальных переменных, вместо того чтобы размещать их в стеке.

Применение данного метода оптимизации часто требует реализации оптимизируемого кода или наиболее критичных его частей на языке Ассемблера, поскольку это позволяет наилучшим образом использовать особенности процессора.

При оптимизации криптографических алгоритмов, в которых часто встречаются повторяющиеся вычисления, организованные как циклы, прибегают к методу развертывания циклов, который заключается в том, что из кода программы ис-

ключаются операторы цикла, а тело цикла явно записывается в коде программы необходимое количество раз.

Такой прием исключает из кода программы операторы цикла, уменьшая, таким образом, время выполнения программы и позволяя более эффективно использовать кэш-память для инструкций.

Еще один подход, применяемый при оптимизации алгоритма шифрования, заключается в том, чтобы выделить из алгоритма части, которые могут быть выполнены на предварительной стадии (инициализация, подготовка ключа шифрования и т.д.), чтобы они не вызывались при каждой вызове функции шифрования. Например, американский стандарт шифрования AES (алгоритм шифрования Rijndael, [7]) предусматривает преобразование ключа шифрования в таблицу внутреннего формата (key schedule) перед тем, как его использовать. Очевидно, что имеет смысл осуществлять подобное преобразование сразу после загрузки ключа в систему и хранить ключ в виде такой таблицы, а не выполнять это преобразование каждый раз при вызове функции шифрования.

Данный подход не универсален, так, в российском стандарте шифрования ГОСТ 28147-89 с ключом не предусмотрено проведения подготовительных процедур, требующих каких-либо вычислений [1].

Несмотря на то, что описанные здесь подходы оптимизации существенно зависят от выбранного криптографического алгоритма, а также от целевого процессора и платформы, их применение может дать существенный выигрыш в производительности.

Определение уровня распараллеливания шифрования

Для систем защиты данных в процессе их хранения можно предложить следующую классификацию уровней распараллеливания шифрования: уровень устройства, когда все запросы чтения/записи для одного устройства шифруются отдельным процессором; уровень запроса, когда отдельный запрос чтения/записи обрабатывается отдельным процессором, и уровень интразапроса, когда каждый запрос чтения/записи обрабатывается несколькими процессорами.

Рассмотрим все достоинства и недостатки предложенных уровней. С точки зрения простоты реализации очевидно, что наиболее простым является уровень устройства, более сложным — уровень запроса и самым сложным — уровень интразапроса.

Тем не менее распараллеливание на уровне устройств даст выигрыш только в том случае, если в рассматриваемой системе есть более одного устройства, на котором реализуется шифрование данных. В случае если такое устройство одно, никакого выигрыша производительности распараллеливание не принесет. Кроме этого, даже если такое устройство не одно, то при наличии в системе числа процессоров большего, чем число защищаемых устройств, оставшаяся часть процессоров останется не задействованной или задействованной не полностью, что не позволит использовать имеющиеся вычислительные мощности наиболее эффективно.

Распараллеливание на уровне запроса имеет те же недостатки, что и распараллеливание на уровне устройства. Причина этого заключается в том, что запро-

сы для большинства существующих устройств обрабатываются по очереди. Это связано с тем, что физический канал ввода/вывода, обеспечивающий обмен информацией с устройством хранения данных (SATA, SCSI, Fiber Channel и т.д.), реализован как последовательный. Таким образом, поступающие запросы на чтение/запись помещаются в очередь, и по одному передаются контроллеру устройства. Кроме этого, логика работы многих приложений при работе с данными, хранящимися во внешней памяти, построена также последовательно — до тех пор, пока приложение не прочитает определенные данные с диска, оно не будет знать, что делать дальше. Это означает, что шифрование данных, передаваемых в отдельном запросе, отдельным процессором также не принесет какого-либо выигрыша производительности, поскольку запросы выполняются последовательно и до полного завершения обработки очередного запроса, включая шифрование, следующий запрос не будет передан устройству хранения данных. Этот вариант в ряде случаев может быть даже хуже, чем распараллеливание на уровне устройства.

Распараллеливание на уровне интразапроса, при котором данные, передаваемые в каждом запросе чтения/записи, шифруются всеми процессорами параллельно, лишено упомянутых недостатков. В этом случае вычислительная мощность, доступная в системе, используется наиболее эффективно вне зависимости от количества защищаемых устройств, числа процессоров и особенностей функционирования контроллера устройств хранения данных.

Выбор криптографического алгоритма

Вопрос выбора криптографического алгоритма является очень важным при проектировании криптоядра. Перечислим наиболее существенные соображения, которые необходимо учитывать при выборе криптографического алгоритма.

Во-первых, необходимо определить класс алгоритма — симметричный или асимметричный. В связи с более высокой скоростью и простотой реализации, а также благодаря отсутствию необходимости распространения ключей шифрования для систем защиты информации используются симметричные криптографические алгоритмы.

Во-вторых, необходимо учитывать криптографическую стойкость алгоритма шифрования. Данный вопрос необходимо решать исходя из степени секретности защищаемых данных. Оценка криптографической стойкости — это довольно сложный вопрос, выходящий за рамки настоящей статьи. В любом случае эксперты в области криптографии и защиты информации рекомендуют выбирать какой-либо широко известный криптографический алгоритм, который хорошо изучен и получил много положительных откликов [5].

В-третьих, для данных типов систем важен тип алгоритма — блочный или поточный. Для блочных алгоритмов единицей обработки информации при зашифровывании и расшифровывании является блок фиксированного размера (как правило, его размеры для большинства алгоритмов составляют 8, 16, 32 и т.д. байта). При этом в настоящее время для всех наиболее распространенных криптографических алгоритмов размер блока в байтах является степенью числа 2. При необходимости зашифровать данные, размер которых не кратен размеру блока,

необходимо дополнять шифруемые данные до размера, кратного блоку (padding, [10]). Это означает, что должна быть отдельная процедура, выполняющая это действие, которая для своего выполнения также требует дополнительных вычислительных ресурсов. В поточных криптографических алгоритмах минимальным размером шифруемых данных является 1 бит, это дает большую гибкость при шифровании различных блоков данных. Однако большинство данных алгоритмов (за исключением ГОСТ 28147-89 в режиме гаммирования с обратной связью) уязвимы к атаке по известному открытому тексту (known-plaintext attack), что необходимо учитывать при реализации. Следует также отметить, что на базе любого блочного алгоритма можно построить поточный с помощью режимов шифрования OFB и CFB, однако обратная трансформация из поточного шифра в блочный не так очевидна. Кроме этого, если в системе защиты реализуется шифрование данных на уровне устройства, чтение и запись на дисковые устройства выполняется также по блокам, с минимальным размером блока 512 байт, или 2^9. При работе с магнитными лентами чтение и запись выполняется также поблочно, и размер блока в байтах также представляет собой степень числа 2. Это означает, что использование блочного алгоритма не потребует какого-либо выравнивания, поскольку размер шифруемых данных всегда будет кратен размеру блока алгоритма шифрования. С учетом вышесказанного для криптоядра имеет смысл выбирать блочный алгоритм или ГОСТ 28147-89 в режиме гаммирования с обратной связью.

В-четвертых, существенную роль при выборе алгоритма шифрования является его быстродействие по сравнению с другими алгоритмами. Очевидно, что при прочих равных для реализации криптоядра необходимо выбрать тот алгоритм, который обеспечивает более высокое быстродействие на целевой платформе с учетом потенциальных возможностей по оптимизации.

В-пятых, необходимо учитывать законодательные ограничения на использование криптографических алгоритмов, поскольку многие государства осуществляют регулирование рынка криптографических систем путем принятия стандартов и ограничения возможностей по использованию криптосистем, им не соответствующих. Например, в РФ, если криптографические средства предназначены для защиты информации в соответствии с законодательством РФ, для защиты конфиденциальной информации в государственных органах РФ, и во многих других случаях, большинство действий в процессе разработки, производства, распространения и эксплуатации СКЗИ должны быть согласованы с ФСБ России [3].

С учетом сказанного для защиты данных в соответствии с законодательством РФ для защиты конфиденциальной информации в государственных органах РФ и в некоторых других случаях, следует использовать алгоритм шифрования, соответствующий ГОСТ 28147-89. В других случаях, когда нет законодательных ограничений, влияющих на выбор, заказчики чаще всего предпочитают алгоритм Rijndael, которому соответствует американский стандарт шифрования AES, с учетом того, что этот алгоритм де-факто включен в состав такого распространенного программного обеспечения как семейство операционных систем Microsoft Windows, Интернет-браузер Internet Explorer и т.д.

Тем не менее, при выборе алгоритма шифрования могут приниматься во внимание какие-то другие, поэтому наиболее универсальным решением будет модульное подключение к системе защиты компонент, реализующих какой-либо конкретный алгоритм шифрования, либо на этапе сборки системы, либо, в виде динамически загружаемых библиотек, на этапе установки или использования. Это позволит учесть конкретные требования, предъявляемые к выбору алгоритма, без значительной модификации всей системы.

Выбор размера блока шифрования

Существенным параметром при проектировании архитектуры криптоядра является размер блока шифрования, поскольку этот размер определяет максимальную гранулярность распараллеливания и непосредственно влияет на результирующую производительность.

В данном случае под размером блока шифрования понимается минимальный размер блока данных, который шифруется с одинаковым набором параметров (ключ шифрования, синхропосылка, и т.д.). Это означает, что если требуется расшифровать какие-то данные, количество которых меньше, чем размер блока шифрования, то сначала необходимо расшифровать весь блок, а потом извлечь оттуда необходимые данные.

Отправной точкой при выборе размера блока шифрования является минимальный размер блока шифрования, характерный для выбранного криптографического алгоритма. Очевидно, что размер блока шифрования в системе защите информации при хранении не может быть меньше, чем размер блока алгоритма шифрования. Это требование выполняется практически во всех случаях, поскольку, как уже говорилось, минимальный размер блока при чтении и записи составляет 512 байт, а размер блока шифрования для большинства криптографических алгоритмов составляет 8 или 16 байт.

Вместе с тем, необходимо знать, каков размер данных, передаваемых в одном запросе чтения или записи, поскольку, для того, чтобы обеспечить эффективное использование вычислительных ресурсов при распараллеливании шифрования на уровне интразапроса, необходимо обеспечить выполнение следующего условия:

$$S_{CRYPT} \le \frac{S_{IO}}{N}.$$

В этой формуле S_{CRYPT} — это размер блока шифрования, S_{IO} — размер блока чтения или записи, N — количество потоков, которые будут одновременно шифровать один запрос.

Для того чтобы оценить размер блока чтения или записи, можно воспользоваться известными параметрами файловой системы. Как известно, различные файловые системы организуют хранение данных таким образом, что минимальным объемом хранимых данных, а, следовательно, и минимальным размером блока чтения или записи, является кластер. В файловых системах FAT16, FAT32 и NTFS, которые поддерживаются операционными системами семейства Windows NT, размер кластера для разделов жестких дисков, размер которых превышает 2 ГБ, составляет от 4 КБ до 64 КБ [3]. При этом надо учитывать, что

для файлов, размер которых составляет хотя бы несколько мегабайт, чтение и запись могут выполняться с применением технологий кэширования и отображения файлов в область памяти, а значит, размер блоков данных, передаваемых в одном запросе чтения и записи, должен быть существенно выше минимального. Для магнитных лент размер блока ввода/вывода должен быть не менее 32 КБ, поскольку, чем больше размер блока ввода/вывода, тем меньше накладные расходы при передаче данных и тем эффективнее работает устройство. Это означает, что, если выбрать размер блока 512 байт, то для восьмипроцессорных серверов даже минимальный размер блока чтения и записи в 4 КБ обеспечит эффективное распараллеливание шифрования на уровне интразапроса. Если же принять во внимание, что размер блока чтения и записи, как правило, больше 4 КБ, то распараллеливание будет эффективно и на системах с большим числом процессоров. С другой стороны, 512 байт — это размер сектора на большинстве дисковых накопителей, и выбор такого размера блока шифрования обеспечит наибольшую простоту реализации.

Особенности использования кэш-памяти

Как уже отмечалось ранее, использование кэш-памяти может существенно сократить время на доступ к данным, хранящимся в основной памяти. В современных вычислительных системах, как правило, используется два уровня кэш-памяти, L1 и L2, которые находятся непосредственно на кристалле центрального процессора. Ниже приведены параметры основной памяти, и кэш-памяти обоих уровней [9] (табл.).

Таблица

Вид памяти	Пропускная способность	Время доступа
Основная память	2 + ГБ/с	50 + нс
Кэш-память L2	10 + ГБ/с	2 + нс
Кэш память L1	50 + ГБ/с	300 + пс

Как видно из таблицы, использование кэш-памяти первого уровня может дать существенный выигрыш в производительности по сравнению с основной памятью, при этом разработчику системы не надо прилагать для этого практически никаких усилий — все необходимые действия по загрузке данных из основной памяти в кэш-память и наоборот выполняются системой автоматически.

Тем не менее как уже упоминалось, алгоритм поддержания когерентности кэш-памяти может негативно повлиять на общую производительность. Проблема может возникнуть, если на целевой платформе для обеспечения когерентности используется политика обновления по записи, и в кэш память какого-либо процессора попала область шифруемых данных, которую обрабатывает другой процессор. В этом случае, при любом изменении данных в процессе шифрования оборудование поддержания когерентности кэш-памяти будет копировать изменяемые данные в кэш-память всех процессоров, в которых находятся копии этих данных, несмотря на то, что этим процессорам копируемая область данных для работы не нужна.

Очевидно, что данная проблема актуальна только для той кэш-памяти, к которой имеют доступ не все процессоры и только в том случае, если размер бло-

ка данных, обрабатываемого одним процессором меньше, чем объем его индивидуальной кэш-памяти.

Для решения данной проблемы предлагается определять размер индивидуальной кэш-памяти процессора, и, если размер блока данных, обрабатываемого одним процессором, будет меньше, чем объем кэш-памяти, то необходимо обеспечить перенос блока данных для каждого процессора в различные области основной памяти. Взаимное расположение этих областей основной памяти должно быть выбрано таким образом, чтобы блоки данных, шифруемых отдельными процессорами, загружались только в кэш-памяти этих процессоров.

При этом необходимо определить, какой алгоритм поддержания когерентности кэш-памяти используется на целевой платформе, и оценить, не будут ли время копирования шифруемых данных в разделенные области памяти превышать суммарную задержку, связанную с обеспечением когерентности кэш-памяти в процессе шифрования этих данных.

Особенности создания и синхронизации потоков

В большинстве современных операционных систем, в том числе в ОС семейства Windows NT, основной единицей исполнения инструкций процессора является поток (thread). Это означает, что для распараллеливания функции шифрования необходимо создать несколько потоков и назначить каждому потоку для шифрования часть блока данных, передаваемых в процессе выполнения одного запроса. Необходимо рассмотреть средства для создания потоков, управления и синхронизации. Важно отметить, что речь идет о потоках режима ядра операционной системы.

Разработчикам драйверов для ОС семейства Windows NT предлагается набор функций для управления потоками [8]. В частности, для создания потоков режима ядра предназначена функция PsCreateSystemThread. Данная функция может вызываться только на самом низком уровне IRQL — PASSIVE_LEVEL, в то время как диспетчерские функции драйвера и функции завершения запроса ввода/вывода могут выполняться на более высоких уровнях IRQL (до DISPATCH_LEVEL, [2]). Кроме этого, функция создания потока реализует выделение памяти и определенную манипуляцию объектами ядра, то есть время ее выполнения может быть весьма значительным. Это означает, что создание потока в процессе обработки чтения/записи не оптимально, это усложняет программный код и ухудшает производительность.

Таким образом, более оптимальным вариантом представляется создание необходимого количества потоков заранее, например, при включении режима прозрачного шифрования для конкретного устройства, использование этих потоков в течение всего времени работы с этим устройством, и их удаление при выключении режима прозрачного шифрования.

Для синхронизации доступа к данным при их обработке в ядре ОС семейства Windows NT могут использоваться упомянутые ранее объекты синхронизации — семафоры, события, мьютексы и т.д. Однако модель обработки данных при распараллеливании шифрования не предусматривает какой-либо синхронизации доступа к данным, поскольку каждый запрос чтения/записи делится на несколько частей и каждая часть независимо обрабатывается своим потоком. Безусловно,

это не означает, что при реализации криптоядра функции синхронизации не нужны вообще, поскольку остаются задачи синхронизации доступа к общим служебным данным, очередям, спискам и т.д. Однако эти задачи не выходят за рамки стандартного проектирования и реализации драйверов, они неоднократно рассматривались в многочисленной литературе, поэтому в данной статье они не обсуждаются.

При координации выполнения потоков, реализующих шифрование, следует иметь в виду, что основной поток (управляющий поток), обслуживающий запрос ввода/вывода, должен запустить несколько потоков, реализующих шифрование (рабочих потоков). Они должны запуститься одновременно, таким образом, при запуске их отношение синхронизации будет «старт—старт» [6]. Для завершения шифрования блока данных, основной поток должен ожидать завершения рабочих потоков, реализуя, таким образом, отношение синхронизации «финиш—финиш», или, как ее еще называют, барьерную синхронизацию. Для реализации координации выполнения потоков также могут быть использованы любые подходящие для этого объекты синхронизации, доступные разработчикам драйверов ОС семейства Windows NT.

Определение количества рабочих потоков

На первый взгляд, задача определения количества рабочих потоков кажется очевидной — их число должно быть равно числу процессоров в системе, или на один меньше, если управляющий поток тоже будет выполнять шифрование части блока данных. В этом случае ресурсы имеющейся вычислительной системы будут использоваться наиболее полно, а выигрыш в производительности, связанный с распараллеливанием — максимальным.

Однако данная задача так просто не решается. Проблема заключается в том, что, во-первых, потоки, выполняющие шифрование, в реальной системе не единственные, кроме них есть другие потоки, которые тоже загружают систему и требуют вычислительных ресурсов. Во-вторых, многозадачность в серверных операционных системах семейства Windows 2000 построена по вытесняющему (pre-emptive) принципу. Это означает, что каждый поток, выполняемый системой, может быть прерван и приостановлен для того, чтобы выполнить поток более высокого приоритета.

В связи с этим разделение шифруемого блока данных на несколько частей и шифрование каждой части в отдельном потоке не гарантирует одновременного выполнения этих потоков на многопроцессорной (многоядерной) архитектуре. Это означает, что потоки, запущенные параллельно, могут выполняться частично параллельно, или даже последовательно, и это может привести не к ускорению операций шифрования, а к их замедлению, поскольку на все операции, связанные с распараллеливанием и синхронизацией, тоже требуются временные затраты.

Из сказанного следует, что важной задачей, возникающей при разработке многопоточного криптоядра для систем с вытесняющей многозадачностью, является задача определения оптимального числа частей, на которые надо разделить блок данных для параллельного шифрования отдельными потоками. Для решения этой задачи можно воспользоваться методами теории массового обслуживания и методами имитационного моделирования.

ЛИТЕРАТУРА

[1] ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. — М.: Госстандарт СССР, 1989.

[2] *Они У.* Использование Microsoft Windows Driver Model. 2-е изд. — СПб.: Питер, 2007.

[3] Приказ Федеральной службы безопасности Российской Федерации от 9 февраля 2005 г. № 66 Об утверждении Положения о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ-2005): Рег. № 6382 от 3 марта 2005 г. // Бюллетень нормативных актов федеральных органов исполнительной власти. Выпуск 11. — М.: Юридическая литература, 2005.

[4] Размер кластера по умолчанию для файловых систем FAT и NTFS [Электронный ресурс] // Центр справки и поддержки Microsoft. — Код статьи: 140365. — Режим доступа: http://support.microsoft.com/kb/140365. — Загл. с экрана.

[5] *Танненбаум Э.* Архитектура компьютера. 5-е изд. — СПб.: Питер, 2007.

[6] *Фергюсон Н.* Практическая криптография / Пер. с англ. — М.: Вильямс, 2005.

[7] *Хьюз К.* Параллельное и распределенное программирование с использованием C++ / Пер. с англ. — М.: Вильямс, 2004.

[7] *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Пер. с англ. — М.: Триумф, 2002.

[8] Federal Information Processing Standards Publication 197. Advanced Encryption Standard (AES) [Electronic resource]. — 2001. — Режим доступа: http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf. — Загл. с экрана.

[9] Process and Thread Manager Routines [Electronic resource] / Microsoft Developer Network: Windows Driver Kit. — Режим доступа: http://msdn.microsoft.com/en-us/library/ms802955.aspx. — Загл. с экрана.

[10] *Shen J.* Modern Processor Design: Fundamentals of Superscalar Processors. — McGraw-Hill, 2004.

PARALLEL CRYPTOKERNEL ARCHITECTURE FOR SERVER STORAGE SECURITY SYSTEMS

A.V. Raevsky

Cybernetics and mechatronics department
Peoples' Friendship University of Russia
Miklukho-Maklaya str., 6, Moscow, Russia, 117198

The article considers design and implementation of parallel cryptokernel for server storage security systems, which protect data at rest. Most important issues of cryptokernel design, such as crypto algorithm optimization, defining the level of parallelization, choosing the crypto algorithm and crypto block size are analyzed. Questions like cache memory usage and thread creation and synchronization are taken into account. Approaches for determining the number of threads working in parallel are proposed.