УДК 004.428

ПОРТИРОВАНИЕ КРИПТОГРАФИЧЕСКИХ БИБЛИОТЕК С ПОМОЩЬЮ SWIG

М. Н. Мицкевич

НИИ прикладных проблем математики и информатики,
Белорусский государственный университет,
Минск, Беларусь, mitskevichmn@bsu.by

Приводятся результаты анализа инструментов высокоуровневого портирования. Описан пример портирования криптографической библиотеки Bee2 на язык C#.

Ключевые слова: портирование; обвязка; генерация кода; нативная функция.

CRYPTOGRAPHIC LIBRARIES PORTING WITH SWIG

M. N. Mitskevich

Research Institute for Applied Problems of Mathematics and Informatics,
Belarusian State University,
Minsk, Belarus, mitskevichmn@bsu.by

Results of the investigation of tools for high-level porting are considered. An example of cryptographic library porting to C# language is described.

Keywords: porting; wrapper; code generation; native function.

1. Введение

Большинство криптографических инструментов написано на языках Си и C++. Не является исключением и реализация стандартизированных в Республике Беларусь криптографических алгоритмов в библиотеке Bee2 [1], которая распространяется под открытой и свободной лицензией Apache License, Version 2.0.

Предыдущий опыт встраивания библиотеки Bee2 в решения, написанные на Java, показал трудоёмкость этого процесса и выявил наличие проблем совместимости. Сейчас актуальной является задача применения белорусской криптографии в решения, написанные на других языках, что явилось причиной проведения анализа инструментов портирования.

Было потрачено много сил и времени на написание, проверку и тестирование исходного кода реализации алгоритмов, поэтому переписывание

с нуля на каждом новом языке программирования изначально не рассматривалось.

Также хотелось покрыть максимально возможное количество языков и платформ, поэтому не рассматривались специализированные библиотеки, рассчитанные на портирование кода под определенный язык программирования, например, ctypes для Python.

Мы хотим получить с минимально возможными усилиями модуль на новом языке программирования (целевом), который будет вызывать функции библиотеки Bee2 привычным для целевого языка способом, используя обвязки для функций языка Си.

На наш взгляд, единственный инструмент, который удовлетворяет заявленным критериям отбора, доступен, развивается и поддерживает большое число целевых языков программирования – SWIG.

2. Обзор SWIG

SWIG (англ. *simplified wrapper and interface generator*) — свободный инструмент для связывания программ и библиотек, написанных на языках C и C++, с интерпретируемыми (Tcl, Perl, Python, Ruby, PHP) или компилируемыми (Java, C#, Scheme, OCaml) языками [2].

Основная цель — обеспечение возможности вызова функций, написанных на C и C++, из кода на других языках для расширения функциональности или повышения производительности.

Программист создаёт файл интерфейсов `.i` с описанием экспортируемых функций, а SWIG генерирует исходный код для склеивания C/C++ и нужного языка. В результате генерируется файл интерфейса для подключения существующей библиотеки или исходный код библиотеки функций-обвязок над функциями C/C++.

SWIG написан на языках C и C++, распространяется по лицензии, похожей на BSD, с февраля 1996 года. Лицензия SWIG позволяет использовать, распространять и модифицировать код SWIG для коммерческих и некоммерческих целей практически без ограничений.

3. Подключение Bee2

Рассмотрим пример вызова функций выработки детерминированной электронной цифровой подписи (ЭЦП) из библиотеки Bee2 в языке C#. Для этого мы должны включить модуль `bign` с функцией выработки ЭЦП, модуль `belt` с функцией хэширования и дополнительные модули для выделения памяти `mem`, для преобразования 16-ричных чисел `hex`, а также модуль `safe`, от которого зависят `bign` и `belt`.

Основной для SWIG файл `bee2.i` начинается с заголовочной секции. В ней прописывается директива `%module`, в которой указано название целевого модуля. Также подключаются модули стандартной библиотеки SWIG с помощью директивы `%include`, например, инструкция `%include "carrays.i"` подключает функции управления массивами.

```
%module bee2
%include "carrays.i"
%include "cdata.i"
%include "cpointer.i"
#ifdef SWIGPYTHON
%include "cstring.i"
#endif
%include "typemaps.i"
```

Далее следует секция включения `%{ ... %}`. Она позволяет включать в код обвязки необходимые для его компиляции заголовочные файлы, а также дополнительные функции, необходимые для корректной работы функций в целевом языке программирования. Например, для выполнения тестов нам потребовалась операция сдвига указателя на некоторое фиксированное смещение, а многие высокоуровневые языки прямо запрещают это делать. Поэтому мы добавили функцию `ptradd`, которая принимает указатель и смещение и возвращает новый смещенный указатель. В итоге секция включения в нашем примере выглядит так:

```
%{
#include "bee2/core/safe.h"
#include "bee2/core/mem.h"
#include "bee2/core/hex.h"

#include "bee2/crypto/belt.h"
#include "bee2/crypto/bign.h"

void* ptradd(void* ptr, int offset) {
 return ptr + offset;
}
%}
```

Далее следует секция объявления типов, массивов и структур, необходимых для того, чтобы SWIG понимал типы аргументов при разборе функций.

```
typedef unsigned int u32;
typedef u32 err_t;
```

```
#define ERR_OK    ((err_t)0)

%array_functions(u32, u32arr)
%apply SWIGTYPE* { size_t* count };
%apply SWIGTYPE* { size_t *ary };
%array_functions(size_t, sizeTarr)

%pointer_cast(octet*, void*, op2vp)
%pointer_cast(void*, octet*, vp2op)
%pointer_cast(unsigned char*, void*, bp2vp)
%pointer_cast(void*, unsigned char*, vp2bp)
%pointer_cast(void*, char*, vp2cp)
```

В самом конце идёт секция разбора. В ней содержатся инструкции `%include`, которые определяют заголовочные файлы, к функциям которых SWIG должен сгенерировать обвязки. Обычно все эти файлы должны быть указаны в секции включения, но для модуля `bee2/core/hex.h` генерируемые по умолчанию интерфейсы были неработоспособны, поэтому вместо подключения модуля пришлось явно объявить функции `hexTo` и `hexFrom`.

Кроме того, тут должны быть объявлены функции, реализация которых находится в секции включения (у нас это функция `ptradd`), чтобы к ним тоже были сгенерированы обвязки, и они стали доступными в целевом языке программирования.

```
%include "../include/bee2/core/safe.h"
%include "../include/bee2/core/mem.h"
%include "../include/bee2/crypto/belt.h"
%include "../include/bee2/crypto/bign.h"

void* ptradd(void* ptr, int offset);

void hexTo(void* dest, const char* src);

#ifdef SWIGPYTHON
%cstring_mutable(char* dest)
void hexFrom(char* dest, const void* src, size_t
count);
#else
%typemap(freearg) char * {  }
%apply SWIGTYPE* { char *dest };
```

```
    void hexFrom(char* dest, const void* src, size_t
count);
    #endif
```

Следует отметить возможность условной компиляции, которая позволяет задавать разные правила генерации обвязок для разных языков программирования. Так, в примере выше видно, что с помощью библиотеки `cstring.i` получается более простое объявление указателя на изменяемую (mutable) строку. Так как эта библиотека SWIG реализована только для языка Python, то в других языках приходится использовать более сложные директивы библиотеки `typemaps.i`.

Также условная компиляция помогает при необходимости добавления функций, необходимых только в определенном языке. Так, например, в языке C# возникла проблема с получением константной строки символов из изменяемой, так как невозможно задать преобразование из типа `char*` в два разных типа C# (`string` и `SWIGTYPE_p_char`). Обойти это получилось, задавая все буферы памяти как тип `void*` и преобразовывая указатели при необходимости либо в строку с помощью конвертера `vp2cp`, либо в массив символов через следующую функцию:

```
#ifdef SWIGCSHARP
%{
char* getStr(void *dest) { return dest; }
%}
%apply SWIGTYPE* { char* };
char* getStr(void *dest);
#endif
```

4. Генерация библиотеки обвязок

В результате компиляции входного файла в SWIG создается готовая библиотека для языка C#, в которой нам важно выделить файл с обвязками `bee2_wrap.c` и главный модуль библиотеки `bee2cs.cs`.

Для всех функций из заданных заголовочных файлов библиотеки генерируется обвязка на стороне Си, реализующая интерфейс, пригодный для импорта в C#, и обвязка на стороне C#, которая скрывает детали реализации импорта функций за интерфейсом, соответствующим исходным функциям библиотеки. Например, для компилятора Си важен тип данных, на которые ссылается указатель.

Для C# также важны логические типы данных. В то время как реализация всех типов указателей одинакова, что учтено интерфейсом импорта нативных функций в C#. Поэтому следующая функция принимает

нетипизированный указатель, затем преобразует его в тип параметра функции `getStr` (также нетипизированный указатель), выполняет с помощью этой функции его преобразование в типизированный указатель на строку символов, затем обратно преобразуется в нетипизированный указатель для передачи в C#, где он будет преобразован в типизированный указатель `SWIGTYPE_p_char`, связанный со строками и позволяющий менять их содержимое.

```
    SWIGEXPORT void * SWIGSTDCALL
CSharp_bcrypto_getStr(void * jarg1) {
      void * jresult ;
      void *arg1 = (void *) 0 ;
      char *result = 0 ;
      arg1 = (void *)jarg1;
      result = (char *)getStr(arg1);
      jresult = (void *)result;
      return jresult;
    }
```

В файл с обвязками `bee2_wrap.c` включаются ещё и дополнительные функции, объявленные в файле `bee2.i`:

```
    char* vp2cp(void* x) {
       return (char*) x;
    }
    char* getStr(void *dest) {
     return dest;
    }
```

Параметр SWIG `-outfile bee2cs.cs` позволяет не разделять код C# на отдельные модули, а собрать весь код в одном файле, в котором содержатся все классы, соответствующие типам Си, а также класс библиотеки и класс нативных функций, о которых скажем отдельно.

В классе нативных функций `bee2PINVOKE` прописаны интерфейсы для сгенерированных нативных функций, выполняющих преобразование имен и типов во внутренние типы и имена библиотеки C#. Для каждой функции прописана точка входа и интерфейс функции, который она получает в C#.

```
[global::System.Runtime.InteropServices.DllImport(
"bee2wrap", EntryPoint="CSharp_bcrypto_getStr")]
public static extern global::System.IntPtr getStr(
    global::System.Runtime.InteropServices.HandleRef
jarg1);
```

По умолчанию имя библиотеки, из которой производится импорт функции, равно имени модуля SWIG. Имя модуля удобно задавать как имя исходной библиотеки, так как для имени класса для представления библиотеки в целевом языке также используется имя модуля. Для того, чтобы переопределить это имя на имя библиотеки с обвязками, используется параметр SWIG `-dllimport bee2wrap`, специфичный только для языка C#.

Основной класс `bee2` содержит набор статических функций, соответствующих функциям из заголовочных файлов, указанных в секции разбора.

В этих функциях вызываются обвязки, реализованные в Си-библиотеке, а также выполняется приведение типов, локализирующее использование нативных указателей внутри класса.

```
   SWIGTYPE_p_char getStr(SWIGTYPE_p_void dest) {
      global::System.IntPtr cPtr =
bee2PINVOKE.getStr(SWIGTYPE_p_void.getCPtr(dest));
      SWIGTYPE_p_char ret =
      (cPtr == global::System.IntPtr.Zero) ? null :
      new SWIGTYPE_p_char(cPtr, false);
      return ret;
   }
```

5. Особенности портирования

В процессе портирования библиотеки Bee2 было отмечено несколько проблем портирования.

1. Преобразование типов данных.

Базовые типы Си хорошо переводятся в типы других языков. Проблемы возникают там, где в самом Си допускается разные варианты как размера типов (как в `size_t`, где размер беззнакового целого зависит от платформы), так и содержимого (как в `char`, который используется и для строк, и для целых чисел).

2. Перенос семантики типов данных.

Рассмотрим следующий пример:

```
typedef unsigned char u8;
typedef u8 octet;
```

При таком введении новых типов через переопределение старых в целевом языке программирования теряется связь между типами, поэтому следует добавлять явные функции преобразования между типами, а также между указателями на эти типы. Кроме того, нужно указать преобразование из нетипизированного указателя в типизированный для каждого типа, который используется в заголовках функций. Например, минимальный набор функций для преобразования однобайтных массивов:

```
%pointer_cast(octet*, void*, op2vp)
%pointer_cast(void*, octet*, vp2op)
%pointer_cast(unsigned char*, void*, bp2vp)
%pointer_cast(void*, unsigned char*, vp2bp)
%pointer_cast(void*, char*, vp2cp)
```

3. Распространение нативных библиотек вместе с C#.

При компиляции библиотеки обвязок мы используем команду, которая размещает библиотеку обвязок в специальном каталоге вида `bee2net/runtimes/$os-$platform/native/$lib.so` библиотеки C#.

Чтобы библиотека обвязок копировалась вместе с библиотекой C# во все проекты, использующие её, нужно в файл проекта библиотеки C# `bee2net.csproj` добавить специальную инструкцию копирования содержимого каталога `runtimes` вместе с библиотекой [3].

Кроме того, нужно добавлять в каждую программу на C# вызов загрузчика, который будет выбирать подходящую библиотеку для каждой платформы.

6. Заключение

SWIG позволяет быстро интегрировать библиотеки Си и C++ во многие языки программирования. Это может быть полезно как для быстрого прототипирования, так и для создания работоспособного макета для полноценного портирования библиотеки.

Предполагается, что описанный подход позволит расширить использование функций защиты информации в программном обеспечении, а также увеличит популярность новых криптоалгоритмов.

Примеры портирования библиотеки Bee2 на C# и другие языки программирования доступны в репозитории `bee2ports` [4].

Библиографические ссылки

1. *Agievich S.* Bee2 // GitHub. URL: https://github.com/agievich/bee2 (date of access: 05.08.2025).

2. SWIG. URL: https://www.swig.org/ (date of access: 05.08.2025).

3. Where to put Native libraries inside a Project and how to reference them? // GitHub. URL: https://github.com/dotnet/runtime/issues/11404 (date of access: 08.08.2025).

4. Bee2ports // GitHub. URL: https://github.com/bcrypto/bee2ports/ (date of access: 11.08.2025).