

CHALMERS



LO! LLVM Obfuscator

An LLVM obfuscator for binary patch generation

Master of Science Thesis

FRANCISCO BLAS IZQUIERDO RIERA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, January 2014

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

LO! LLVM Obfuscator

An LLVM obfuscator for binary patch generation

FRANCISCO BLAS IZQUIERDO RIERA

©FRANCISCO BLAS IZQUIERDO RIERA, January 2014.

Examiner: ANDREI SABELFELD

Supervisor: JONAS MAGAZINIUS

Chalmers University of Technology

University of Gothenburg

Department of Computer Science and Engineering

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Cover:

The dragon representing the LLVM project having a rusty part of him replaced by a new blurry part.

The picture represents the idea of replacing broken parts of projects with new obfuscated parts.

Department of Computer Science and Engineering

Göteborg, Sweden January 2014

Abstract

As part of this Master's Thesis some patches to LLVM have been written allowing the application of obfuscation techniques to the LLVM IR. These patches allow both obfuscation and polymorphism which results in code that is both hard to read and different from previous versions. This, makes finding the real changes made between versions harder for the attacker.

The techniques are applied using a function attribute as the seed for the CPRNGs used by the obfuscation transformations as a source of entropy. As a result it is possible to mark the functions that should be obfuscated in the prototypes allowing the developer to create binaries with the desired amount of changes and a sufficiently large amount of functions that are hard to read and (if the seed is changed) different from previous versions.

In this Master's Thesis the possible ways in which the applied techniques can be "reversed" have been evaluated to be able to compare the resulting code. For this to succeed a transformation able to obtain LLVM IR from the resulting binary code is necessary, this was not done as part of this work.

Acknowledgements

Life is all about choices: you exist because at some point of time your parents made a choice on having a child¹. You are probably reading this text because you have made a choice on that it will be interesting and you likely are who you are because others have influenced your life through their own choices along with yours.

Choices can be good or bad with a whole gamut of grays in the middle. But independently of the result, the path that a choice makes you take is more important and enriching than the choice itself.

Despite I'm the one presenting this work as my Master's Thesis and thus closing a chapter of my life, it wouldn't have been possible for me to do so if some people hadn't chosen to support me in one or another way. This pages will never be enough to thank all of them.

Even worse, though, was making the big mistake of not writing them as I did on my Computer Engineer final project [11], to make up for this, this section will also cover the acknowledgments that weren't done in that publication.

First of all and typical as it may seem I'd like to thank my parents. If they hadn't chosen to have me this thesis nor the project would have existed. Transitively the thanks should expand to their parents and those's parents (and so until the first freewilled being I suppose) for taking similar choices.

Next of course comes the rest of my family, they have chosen to support me in my studies all along and without their help this wouldn't have been possible.

Going back before I even started my project some people I should thank would be Jon Ander Gómez who had arranged the ICPC local programming contests in the UPV as they helped me develop the skills I needed later and Miguel Sánchez and Alberto Conejero who have been amongst the best teachers I had. Also I should thank the ELP group at the DSIC department for giving me a chance of tasting what research felt like back then.

Focusing on my Computer Engineer project I should be thanking Pedro López who pushed for me, Julio Sahuquillo, my examiner at the UPV, Per Stenström, my examiner at Chalmers and Rubén González, my advisor and the biggest influence in the project.

Finally focusing on this actual work I'd have to thank the PaX Team and Anthony (blueness) G. Basile for their great input on the project, Jonas Magazinius for agreeing to be my advisor (and putting up with me all this time), Andrei Sabelfeld for being such a nice examiner and Grim Schjetne for being such a nice opponent despite so little

¹Or just on not using contraceptives that fateful night and then following with the pregnancy.

notice. Also thanks to all who attended the presentation and provided input which has helped improve this document.

But specially, thanks to all of you who hasn't been mentioned on this page, your small contributions are what really made this possible.

Whilst doing this work many things have changed in my life, I have seen the Hackerspace at Göteborg where I'm writing this lines take of, I have started a relationship with a girl, and have met some new friends. I don't know what the future will bring with it as it's really hard to see it now, but I'm quite convinced on what the past has brought thanks to all that people, as this future yet to come wouldn't have been possible without their incredible help.

Thus, to all those who have helped in one way or another to make this possible I can only say ¡Muchísimas gracias desde lo más profundo de mi corazón!²

Francisco, Göteborg 14/3/2013

²Thanks a lot from the bottom of my heart!

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem statement	3
1.2.1	Goals	3
1.2.2	Delimitations	3
1.3	Existing research	5
1.4	Existing solutions	6
1.5	Thesis Structure	7
2	Method and Development	8
2.1	Information gathering	8
2.2	Development	9
3	Definitions	10
3.1	Compiler	10
3.1.1	Intermediate representation	10
3.1.1.1	SSA	11
3.1.1.2	PHI Node	11
3.1.1.3	Basic Block	11
3.1.2	Frontend	11
3.1.3	Middle end	11
3.1.4	Backend	12
3.2	LLVM	13
3.2.1	Clang	13
3.2.2	Dragonegg	13
3.2.2.1	LLVM IR	13
3.2.2.2	Evaluation pass	14
3.2.2.3	Transformation pass	14
3.2.3	DAG	14
3.2.3.1	Legalization	14

3.2.3.2	Register allocation	14
3.3	Code obfuscation	15
3.3.1	Obfuscation transformations	15
3.3.1.1	Control Flattening	15
3.3.1.2	Constant Obfuscation	15
3.3.1.3	Opaque predicate	16
3.3.2	Polymorphism transformations	16
3.3.2.1	Register swap	16
3.3.2.2	Dead code insertion	16
3.3.2.3	Code reordering	16
3.4	PRNG	17
3.4.1	CPRNG	17
3.5	Encryption algorithm	19
3.5.1	Symmetric key encryption algorithm	19
3.5.2	Block encryption algorithm	19
3.5.3	AES	20
3.5.3.1	CTR mode of operation	20
3.5.3.2	CMAC mode of operation	21
4	Algorithm Implementation	22
4.1	Control Flattening	22
4.2	Constant obfuscation	24
4.3	Register Swap	26
4.4	Code reordering	27
4.4.1	Instruction reordering	27
4.4.2	Basic block reordering	28
4.5	CPRNG	29
5	Code design considerations	30
5.1	Coding conventions	30
5.1.1	Transformation specific conventions	30
5.1.2	Auxiliar library conventions	31
5.2	Design choices	32
5.2.1	AES implementation used	32
5.2.2	LLVM Transformations	32
5.2.2.1	Transformation parameters	32
5.2.2.2	Transformation implementation	32
6	Transformation implementations	33
6.1	Obfuscation key	33
6.1.1	addmodulekey	33
6.1.2	propagatmodulekey	33
6.2	Obfuscation	34
6.2.1	flattencontrol	34

6.2.2	obfuscateconstants	36
6.3	Polymorphic	39
6.3.1	bbsplit	39
6.3.2	randbb	39
6.3.3	randins	40
6.3.4	randfun	41
6.3.5	randglb	42
6.3.6	swapops	42
7	Evaluation	44
7.1	Proof: the CPRNG is a good PRNG	44
7.1.1	Determinism	44
7.1.2	Uniformity	44
7.1.3	Independence	45
7.1.4	Function period	45
7.2	Proof: the CPRNG is secure	46
7.2.1	Next bit-test resistance	46
7.2.2	Impossibility to know the result if only the state is known	46
7.2.3	State compromise extension resistance	46
7.3	Proof: the key derivation is secure	47
7.4	Reversing the transformations	48
7.4.1	Defining a global ordering of values	48
7.4.2	Defining a global ordering of instructions	48
7.4.3	Reversing the randfun and randglb transformations	49
7.4.4	Reversing the swapops transformation	49
7.4.5	Reversing the randins transformation	49
7.4.6	Reversing the obfuscateconstants transformation	49
7.4.7	Reversing the flattencontrol transformation	49
7.4.8	Reversing the bbsplit transformation	50
7.4.9	Reversing the randbb transformation	50
7.5	Binary patch obfuscation technique evaluation	51
8	Conclusions	52
9	Future development	53
	Bibliography	54
A	Using the tools	58
B	Popularization	61
B.1	Programming computers	61
B.1.1	Assembly	62
B.1.2	Programming languages	63

B.2	Compilers	64
B.3	Antireverse engineering	65
B.3.1	Obfuscation techniques	65
B.3.1.1	Control flattening	66
B.3.1.2	Constant obfuscation	68
B.3.1.3	Register swap	69
B.3.1.4	Instruction Reordering	70
B.3.2	Focused obfuscation	71
B.3.3	Compiler-level obfuscation	72
B.4	Deobfuscation	73
B.4.1	Control unflattening	73
B.4.2	Constant deobfuscation	73
B.4.3	Register swap	73
B.4.4	Instruction Reordering	73
B.5	Program updates	74
B.6	Practical example	75
C	Source code	76
C.1	Patches to LLVM	76
C.2	Patches to Clang	77
C.3	Obf library	79
C.4	AES library	108

1

Introduction

1.1 Background

As stated by [5], there is no silver bullet to prevent programmers from making mistakes when coding applications. Some of these errors may not necessarily cause more than nuisance when hit by the users of the software, but some of them may actually end up being vulnerabilities exploitable by a third party which can have undesirable effects for the software user ranging from the lack of the availability of said software to more serious compromises like letting the attacker take control of the system.

Generally the likelihood of a software error increases with the size and complexity of the software, likewise the probability of one of the errors being exploitable increases with the amount of software errors. The situation is worsened as in the current world the majority of devices use a reduced set of software programs as some softwares are quite large and complex. For example, according to StatCounter more than half of the OS used to browse the web are Microsoft Windows NT derivatives [24] (mostly Windows 7 and XP) and two thirds of the web browsers used are either Microsoft Internet Explorer, Google Chrome or Mozilla Firefox [25]. This use of a reduced set of software programs is not only because of the size and complexity of these but also because of the incompatibilities between them. As a result of this lack of diversity, vulnerabilities can affect really large amounts of devices and, since most are connected to the Internet, attackers can remotely exploit these.

Usually, when a vulnerability is discovered or reported to the creator of the affected software, he addresses the issue and creates a new version of said software where the problem has been fixed.

Since distributing a whole copy of the new version of the program may require many resources, for example the Firefox 27.0.1 xul.dll file containing most of the GUI runtime is 21.7 MB, most of the times the developer releases instead an small file containing the required changes made to the software in order to address the issue at hand and maybe a small program able to apply these changes to the software. The files with the required changes are generally known as patches and the process of applying them is known as patching although this word may also be used to refer to the whole process of fixing the issue. This process, may also be used in order to address other type of software errors (known as bugs) and to add new features that improve the software, although the latter is quite rare.

When a developer deploys an upgrade, some time has to pass until all the users have actually applied the upgrade and their systems are secure from the attack. This will be the case even if the update process may be done automatically by the software itself and without user intervention. This time window from the time the patch is made public to the time the last user applies the patch to fix the software could be used to attack said system by third parties inferring the vulnerability being fixed from the patches being published.

1.2 Problem statement

As patches tend to be small to reduce the amount of resources required by the process, it becomes easy for the attacker to know what has been changed on the system and via reverse engineering discover the fault being fixed and, if the user hasn't patched the system yet, abuse it. This is usually known as a 1-day vulnerability [22].

1.2.1 Goals

The goal of this thesis is providing a set of tools that can help the software developer increase the amount of time required by the attackers to find the particular changes being done by the patch and also increase the time required for the attacker to understand them. Two different methods will be combined to do so:

1. Applying obfuscation transformations to the code in order to increase the difficulty of following the resulting code by the attacker.
2. Applying polymorphism transformations in order to increase the amount of differences in the patch and, thus, decreasing the signal to noise ratio.

The project aims are to:

1. Provide a set of tools allowing the application of those transformations without modifying the original code during the compilation of said files.
2. Adapt the polymorphism and obfuscation transformations so they can be applied to a compilers intermediate representation. In particular to LLVM's IR which is three way SSA.
3. Create a PoC integrating these tools into a real compiler. Allowing the developer to focus the transformations only on the desired functions.
4. Evaluate the effectiveness of the transformations and, if possible, explain the steps that could be taken to reverse them or at least make them useless.

1.2.2 Delimitations

Any project, comes with certain limitations to its goals as they only have a particular amount of resources available for their completion. In this project, given the amount of time available for this project and the resources available for its completion, the following limits will apply to its goals:

1. Not all the possible transformations will be adapted nor evaluated.

2. A proof of concept for the ways in which the attacker could make the transformations useless, if found, will not be developed.

1.3 Existing research

QUITE a lot of research has already been done on the topics of code offuscation despite being proved that some functions can't be obfuscated at [2]. One of the most relevant papers is [32] which inspired most of the ideas used later in [33] and in this thesis. Also [15] provides other approaches towards obfuscation to prevent simple dissassembling.

The efectiveness of these techniques has been analized on papers like [21] which also contains an overview on some of them. Also some research on reversing these techniques has been already done as stated by [28], [20] and [19].

Since obfuscation allows hiding the intentions of the code being executed is no surprise that one of the main focus of obfuscation has been it's use on Malware programs as shown by papers like [4] or [26]. Despite the techniques explained there are useful for the work done here the project's focus is very different.

Going into more recent research, a quite good review on the obfuscation techniques available can be found on [34]. Also One of the most recent and promising developments in this field is the technique pointed in [7]. Finally, some development on obfuscation using LLVM is presented on [14, 6, 26, 23].

1.4 Existing solutions

WHILST many solutions do exist to allow code obfuscation, none allows focusing the transformations on the desired functions by marking them thus allowing the developer to control the size of the final patch. Also, many tools just focus on obfuscating the resulting binary code so a new tool needs to be developed for each architecture supported. In this project the focus will be on the intermediate representation code resulting in a more portable tool.

The most similar project to the one being done is `obfuscator-llvm` [13]¹. Also the project comes with some limitations, namely: the need of identifying functions by name which is not adequate in some languages (like C++) where mangling is used; the use of a CPRNG seeded randomly which will result in different code on every compilation making patch generation harder; and a control flattening implementation which heavily depends on memory accesses which would need later optimization.

¹Code is available at <https://github.com/obfuscator-llvm/obfuscator>

1.5 Thesis Structure

THIS thesis starts with an Introduction section where the Background behind the project is presented, the problem being solved, the project goals, the limits set on them, the existing solutions and the structure of this work.

Afterwards, it proceeds with a Method and Development section explaining how the gathering of information was done and how the project was developed.

The next section: Technical Descriptions and Definitions introduces the different concepts and terms used on this work and can serve as an introduction for a less experienced researcher.

In the Implementation section the algorithms implemented in this project (along with the CPRNG) are explained and how they were converted into code.

In the Evaluation section proof is provided of the validity of the CPRNG and the key generation system used with them. It's also proved how an attacker could reverse the developed transformations and evaluate the technique used for obfuscation of binary patches.

Finally the Conclusions and expectations on Future Developments based on the project's code are presented.

References along with an annexes with instructions on how to use the tool, another with an explanation of the project aimed to the general public and the project sources and patches are provided.

2

Method and Development

2.1 Information gathering

INFORMATION gathering has been done mainly by looking for papers and websites with the desired keywords on Internet search engines and then looking also on their sources to find other interesting material related to the issue in question.

The main focus has been finding obfuscation transformations that could be used on this project, for this [1] and [33] have been of special interest.

2.2 Development

FOR this work the LLVM compiler and the Clang frontend were chosen because of the quality of their documentation (specially by providing quite explicative tutorials at [18] and [27]).

As a result the language used when developing the system was C++ as that's the language used by these two projects. Although some parts of the code were written in C, in particular the AES library providing the required AES modes for the CPRNG which is derived from Gladman's library [9].

Since one of the objectives of the project is producing code that can be merged to LLVM in the future, development was made against the svn sources for the project, the case for clang is similar¹.

The development platform was a Gentoo Hardened installation based mainly on stable sources but including some unstable packages.

¹Anyways, the patches to the sources are all available for reference at http://klondike.es/programas/llvm_obf/

3

Definitions

3.1 Compiler

A compiler is a tool able to perform translations from a language to another. Generally from a higher level language to a lower level language which is nearer to the one of the destination platform.

When creating a compiler two options can be chosen, doing direct translation between each source and destination or using a frontend to convert the language to a common intermediate language and then a backend to convert from this common language to the destination language.

The first alternative, allows to exploit the most of the expressibility of the destination language and results in more efficient and smaller translations but requires a different compiler for each source and destination pair which means you'll need the number of source languages times the number of destination languages different compilers to cover a particular set of source and destination languages.

The second alternative, allows to keep a common pipeline to optimize the resulting intermediate language and may result in larger and slightly less efficient translations. This alternative only requires to have as many frontends as source languages and as many backends as destination languages.

3.1.1 Intermediate representation

This name is used to refer to the language or set of languages to which the compiler will translate the original code before doing the translation into the desired final

language. It is also used to refer to the code written in any of these languages.

Generally, the generic optimization transformations are performed here as they can apply to all the destination languages.

IR is the abbreviation of intermediate representation and is used by LLVM to refer to the language used by them in the optimization stages.

3.1.1.1 SSA

SSA is an acronym for static single assignment form. In general it is used to refer to a property held by intermediate representations: having each variable assigned only one time.

3.1.1.2 PHI Node

PHI Nodes are used by SSA languages to assign a variable with the appropriate value depending on the code block from which the node is reached. This allows SSA languages having variable's values assigned from other blocks when more than one block jumps to a particular block, as happens for example with loops and conditional structures.

3.1.1.3 Basic Block

A basic block or BB is a set of instructions with a single entry point and a single exit point. As a result, jumps cannot point to the middle of a basic block and basic blocks can't contain more than one jump instruction which is always placed at the end.

3.1.2 Frontend

The frontend is the part of the compiler transforming the original source language into the first intermediate representation used by the compiler pipeline.

Amongst the responsibilities of this part of the compiler are checking the validity of the source code detecting and warning the user of the found errors and doing any source language specific optimizations and extracting the metadata from the source code for usage at later stages.

3.1.3 Middle end

This is the part of the compiler that handles the transformations between intermediate representations and the transformations to code written on each of them.

The main responsibility of this part of the compiler is optimizing the code returned by the frontend.

3.1.4 Backend

This is the final part of the compiler pipeline which transforms the last intermediate representation in the pipeline into the desired destination language.

The main responsibilities of this part of the compiler are legalizing the code applying transformations so it only uses the instructions supported by the target architecture, doing target specific optimizations, allocating the source architecture registers to the different instructions and emitting them.

3.2 LLVM

LLVM is a compiler framework with various native frontends (amongst others, C, C++, objective C, opencl and Haskell) a gcc intermediate representation code frontend (allowing support for Ada, D and Fortran), and backends for many platforms both CPU and GPU based (including ARM, x86, x86-64, MIPS, Nvidia's PTX and ATI's R600).

LLVM uses the frontend to transform the source code into an SSA intermediate representations known as IR, runs the desired optimization and transformation passes over this code and then converts the SSA in to a DAG that is passed to the backend for legalization, register assignation and instruction emission.

A good explanation of the inner workings of the LLVM pipeline can be found at [3]

3.2.1 Clang

Clang is a frontend for LLVM supporting C-type languages (C, C++, Objective C, OpenCL, etc.). It is the frontend used by default when compiling those languages by the LLVM compiler.

3.2.2 Dragonegg

Dragonegg is a frontend that allows using most of the gcc frontends with LLVM, it allows support of languages like ADA or Fortran.

3.2.2.1 LLVM IR

LLVM IR is the SSA language used internally by LLVM for intermediate representations.

This language can be represented by a set of memory structures during compilation time, bytecode when storing it on a disk or passing it around pipes and a human readable assembly like representation useful for developers.

Well formed code using this language must hold at least the properties stated by the Verifier pass.

A good description of the language can be found at [17] and the properties held by well formed language instances are explained in the Verifier.cpp file.

3.2.2.2 Evaluation pass

An evaluation pass parses the IR to generate some information about the code that can be used by other passes without modifying the code.

3.2.2.3 Transformation pass

A transformation pass parses the IR and generates a new IR and maybe some information about the generated IR. In general theses passes should generate a functionally equivalent IR in order to be considered correct.

This is the type of the optimization passes used by LLVM and also of the obfuscation passes that have been generated on this project.

3.2.3 DAG

Dag stands for directed acyclic graph. DAGs are the IR passed to the backends for their transformation into the target instructions.

3.2.3.1 Legalization

Legalization is the process by which the backend transform not supported instructions in the DAG into instructions supported by the instruction emitter.

For example, this stage would convert floating point instructions in an architecture into calls to auxiliary functions supporting them or sets of integer instructions able to do the same operations.

3.2.3.2 Register allocation

This is the process by which the registers available in the target architecture are assigned to be source and destination registers for the DAG instructions. Performing this process properly will result in significant performance differences in the resulting code, specially for architectures with a small number of general purpose registers.

Finding the optimal allocation for registers can be reduced to graph coloring which is known to be an NP-Complete problem although a proof exists in [10] proving that it can be done in polynomial time when using SSA form.

3.3 Code obfuscation

CODE obfuscation is the procedure by which the input source code is transformed into a harder to read code which is functionally equivalent to the source code. Unlike optimization transformations, obfuscation ones don't necessarily produce faster code as they only focus in making the resulting code harder to understand by humans. For example the control flattening code reduces the efficiency of the code as it makes the job of the jump predictor harder, in a similar way different constant obfuscation techniques make the processor execute more instructions to get the same results.

3.3.1 Obfuscation transformations

Obfuscation transformations are transformations which perform code obfuscation. The term will be used to refer only to those code obfuscation transformations which aren't intended to generate polymorphic code like constant obfuscation and control flattening despite this term generally covers a wide variety of transformations intended to make the resulting code harder to read.

3.3.1.1 Control Flattening

The control flattening transformation was initially defined by Wang on [32] The general idea behind it is picking two or more basic blocks and make them jump unconditionally to a new basic block where the destination block will be chosen depending on the previous block. In the project, the transformation is applied to all the basic blocks inside a function resulting in a single main block which decides where to jump afterwards. The details of the algorithm implementing this transformation with LLVM IR will be explained in the implementation section.

3.3.1.2 Constant Obfuscation

The constant obfuscation transformation intends to make constants harder to read by transforming them into a set of instructions which will produce the desired constant. There are some ways of doing so:

1. Fetching the constants from a memory position, for example an array, Wang proposed on [32] using aliasing transformations over the array to make reversing the transformation hard (although this last part hasn't been implemented in this particular project).
2. Encrypting the constants, for example with arithmetic operations, converting them into a cipher text and a key which when combined will result into the desired constant.

3. Using an opaque predicate which will result on the desired constant. This was not implemented for this project either.

3.3.1.3 Opaque predicate

An opaque predicate is a predicate that will return the same value independently of the input values. These are usually derived from mathematical identities.

3.3.2 Polymorphism transformations

These transformations are intended to pick one of many possible transformations of a type of the source code.

By changing the seed of the CPRNG used by these transformations you end up with different instances of code functionally equivalent to the source code, which can be helpful in making the set of differences of a patch larger.

3.3.2.1 Register swap

This transformation works by changing one general purpose register by another on the code. This results on different instructions depending on the architectural register being used.

3.3.2.2 Dead code insertion

Dead code insertion consists on inserting code that is unused by the resulting program. This dead code may even be executed by the program but it's results not used by it.

3.3.2.3 Code reordering

Code reordering reorders the code inside a program to result in multiple different programs depending on how the code is reordered.

3.4 PRNG

A PRNG or PseudoRandom Number Generator is a piece of code used to generate a series of number which has properties similar to those of real random numbers. Since PRNGs generate the pseudorandom numbers by keeping a state derived from an initial seed (which is a small number used to initialize it) it should be noted that they are deterministic as they will generate the same sequence when given the same seed and thus produce reproducible results.

Good PRNGSs follow these properties [8].

Determinism Given the same seed the PRNG will produce the same sequence of numbers.

Uniformity In the sequence all numbers are equally probable

Independence Each output doesn't appear to depend on previous ones

The previous properties in turn cause these properties too:

Good distribution The outputs are evenly distributed along the sequence

Good dimensional distribution The outputs are evenly distributed when analyzed over many dimensions

Appropriate distance between values The distance between the values generated is similar to that of a real random number generator

Long period The generator requires a large amount of iterations before ending up in the same state (and thus generating the same sequence again).

An example of PRNG is the `rand()` function provided by the C library (seeded by the `srand()` function).

3.4.1 CPRNG

A CPRNG for Cryptographic PseudoRandom Number Generator or CSPRNG for Cryptographically Secure PseudoRandom Number Generator is a PRNG with some added properties which make it more resistant to cryptanalysis.

CPRNGs satisfy the properties of a good PRNG but also the following properties which are stronger:

Satisfy the next-bit test given the first k bits of the sequence there is no polynomial time algorithm able to predict the next bit with more than 50 accuracy.

Withstand state compromise extensions if the state of the CPRNG, or part of it, has been compromised it should be impossible to guess the previous values returned by the generator.

3.5 Encryption algorithm

AN encryption algorithm is an algorithm which given some data and a key merges the data with the key in a way that people not knowing the key can't read the data being encrypted with the algorithm.

Although there are some non standard hieroglyphs carved in Egypt ca 1900 BC, these are suspected to be just written as entertainment for other literate people. As a result, the first verified cryptography use dates to 1500 BC when an encrypted Mesopotamian clay tablet containing some recipes considered trade secrets were written.

Despite this, not much serious work on cryptography and cryptanalysis was done until very recently when computers where used to automate the processes.

There is an encryption algorithm (one-time pad) which when used correctly is unbreakable as the entropy provided by the key (if truly random) is equal to the entropy of the message thus making it impossible to derive any information.

Old encryption algorithms like Caesar or Vigénere ciphers. Along with DES, AES, RSA, or RC4 are examples of such algorithms.

3.5.1 Symmetric key encryption algorithm

A symmetric key encryption algorithm is an encryption algorithm which has the same key used for encryption and decryption of data so the key has to be protected from outsiders to protect the data.

Symmetric key encryption algorithms are the opposite of asymmetric key encryption algorithms where data is encrypted with a key and decrypted with a different one (with no easily computable way of getting from one key to the other), thus the encryption key can be published and is called a public key whilst the decryption key is kept secret by the receiver of the message and is called a private key.

The classical ciphers, and others like DES, AES and RC4 are symmetric key encryption algorithms.

3.5.2 Block encryption algorithm

A block encryption algorithm is an encryption algorithm which works over fixed size groups of bits independently.

Generally these can be considered a pseudo random permutation, that is a function doing a 1 to 1 mapping of an n bit input into an n bit output where the key is used to choose one of the possible mappings.

As a cipher able only to encrypt a particular block size is quite unusable as is various modes have been developed for these types of ciphers which make their use easier. ECB where the message is divided in blocks and then encrypted with the same key is the simplest but is also quite insecure as equal blocks will be encrypted into the same output.

Because of this, more advanced modes like CBC (where the previous result is mixed with the plaintext before encryption) are available. Also advanced modes to convert block encryption algorithms into stream ciphers (which encrypt single bits), authentication ciphers or even provide authenticated encryption (with or without authenticated data) amongst others exist. The security of most of these are usually based in the assumption of the algorithm being a pseudorandom permutation.

Examples of such ciphers are AES or DES.

3.5.3 AES

AES for Advanced Encryption Standard, is the NIST standardized version of Rijndael, the winner of the AES selection process. It is a symmetric key block encryption algorithm with a block size of 128 bits and key sizes of 128, 192 and 256 bits.

The AES competition was held to choose an appropriate successor to DES the previously NIST standardized symmetric key block encryption algorithm which had key sizes of 56 bits and block sizes of 64 which could be attacked by brute force.

Thanks to its standardization and widespread usage efficient free software implementations exist along with very efficient hardware implementations, including the ones on newer x86 processors.

3.5.3.1 CTR mode of operation

The CTR mode of operation converts a block cipher into a stream cipher by making it encrypt blocks which contain an increasing counter and then xoring the result with the plaintext as would be done by any stream cipher. This provides some advantages as it allows easy parallelization of the algorithm when applying it to various blocks and as with any stream cipher padding is not needed.

The CTR counter can be implemented in many ways (for example by multiplying a non zero block number by a prime) but the method generally chosen is applying an increment of 1 to the unsigned integer number of as many bits as the block size represented by the previous block as it is simple enough (specially when using a carry aware addition) and still secure.

3.5.3.2 CMAC mode of operation

The CMAC mode of operation is an authentication mode which generates an authentication tag for a given input. This mode of operation is not very different from how a keyed hash based authentication algorithm would work.

When used with a secret key the CMAC mode will prevent any information on the message to be derived from the authentication tag as long as the key is not known and the cypher is secure. But if used with a known key then it can be reversed in some cases by the method shown in [12] but still it will act as a simple entropy summarization algorithm which is still useful for key derivation from variably sized data.

4

Algorithm Implementation

4.1 Control Flattening

CONTROL flattening tries to ensure all basic blocks end up onto the same block where the choice for the next block will be done based on the information provided by the previous block (including which block that was) in this case, we are implementing this transformation with the following algorithm:

1. Create if needed an entry block containing only a non conditional branch instruction to the next block.
2. Create if needed an unreachable block.
3. Create the main block which will do the jumps
4. For each instruction referring to instructions on other blocks (or to the same block if the instruction is a terminator instruction which can't be transformed) create a phi node on the main block which will keep the operand value and replace the use by this new phi node instead. Liveness information can be used to keep only the values in the PHI whilst the variable is live.
5. Move all the phi blocks to the main block as it will be the only place where the source of jumps can be known.
6. Split the blocks who have a terminator not handled by the algorithm
7. Assign a block ID to every block we can jump to from the main block.
8. Create a PHI node (called main PHI) on the main block which will chose the destination block (here, this is an ID) depending on the source of the jump.

9. Apply the required transformations depending on the type of terminator block so the jump will work as expected. As of now the only defined is replacing conditional jumps by a select choosing the appropriate block ID depending on the condition value and unconditional jumps by an unconditional jump to the main block. This step also adds the appropriate value to the main PHI.

10. Create the switch instruction in the main block jumping to the appropriate block depending on the block id contained by the main PHI.

In order to improve the obfuscation generated by this transform, a pass which randomly splits BBs can be used to make the BBs smaller and thus harder to follow.

4.2 Constant obfuscation

CONSTANT obfuscation is implemented by replacing constants by a set of instructions resulting in such a constant. This can only be done when the instruction having the constant replaced can use the result of other instructions instead of a constant in that position. The algorithm used is as follows:

1. Generate a pointer to the array containing constants converted into memory fetchs.
2. For each basic block on each function:
 - 2.1. For each phi instruction using a constant run the constant obfuscation transform inserting new instructions before the terminator instruction of the source block which causes the constant to be chosen.
 - 2.2. For each non PHI instruction using a constant which can be replaced by the result of an instruction run the constant obfuscation transform inserting new instructions before the instruction being processed.
 - 2.3. Generate the array containing the constants moved to memory and make the pointer point to it.

The algorithm to obfuscate constants is as follows (as of now it is only implemented for integer constants as these have predictable arithmetic):

1. Choose one of the following methods randomly if the preconditions are met:
 - 1.1. Memory fetch. Preconditions: the integer is smaller or equal in size to the integers in the array containing integer constants.
 - 1.1.1. Create a constant with the index of the constant in the constant array.
 - 1.1.2. Push the integer value to the list of constants of the constant array.
 - 1.1.3. If desired, reobfuscate the constant containing the array index.
 - 1.1.4. Add an instruction to load the array address from the array pointer.
 - 1.1.5. Add an instruction to calculate the displacement of the constant in the array from the loaded address and the index.
 - 1.1.6. Add an instruction to load the desired constant from the resulting address.
 - 1.2. Equivalent arithmetic instructions. Preconditions: none
 - 1.2.1. Generate a random constant, C1.
 - 1.2.2. If desired, reobfuscate C1 with half of the probability.
 - 1.2.3. Choose randomly a reversible binary operation (as of now these are add, sub and xor)

1.2.4. Generate a constant, C2, which when applied with the binary operation to C1 will result in the original constant.

1.2.5. If desired, reobfuscate C2 with half of the probability.

1.2.6. Insert an instruction applying the binary operand to C1 and C2.

4.3 Register Swap

SINCE this is heavily architecture dependent, a small portion of this functionality is defined by randomly swapping the operands of binary operators where this operation is possible. The effect is improved later by the code reordering transformation.

4.4 Code reordering

CODE reordering inside functions is mainly implemented by randomly reordering the basic blocks and the instructions inside the function. The algorithm has some peculiarities in each case defined in the following sections.

4.4.1 Instruction reordering

Instruction reordering requires instructions with side effects being executed always in the same order (as their effects can cause hidden dependencies). It also needs that dependencies are executed before their uses. The reordering is applied on a per basic block basis to reduce the scope of the pass as things would get more complicated when jumps are involved.

The algorithm starts by randomly reordering the PHI instructions inside the basic block as these have the only dependency of being the first in the block.

The algorithm then generates the dependency map as follows:

1. For each instruction in the BB after the normal instruction insertion point not being the terminator:
 - 1.1. If the instruction is already on the dependency map fetch the current dependency list.
 - 1.2. If the instruction may have side effects add it as a dependency of any future instruction reading from memory or with possible side effects.
 - 1.3. If the instruction fetches data from memory add it as a dependency of any posterior instruction which may have side effects.
 - 1.4. For each operand of the instruction which is an instruction result, on the same block and which is different from itself or a PHI instruction:
 - 1.4.1. If the dependency list is still undefined create it.
 - 1.4.2. Insert the operand in the dependency list.
 - 1.5. If the instruction has no dependency list then add it to the candidate list.

With the dependency map and the initial candidate list the algorithm picks then an instruction randomly from the candidate lists and places it at the current insertion point, once it does so it removes all the dependencies of that instruction from the dependency list and moves instructions with an empty dependency list to the candidate list.

4.4.2 Basic block reordering

Basic block reordering only requires that the entry block is kept the same. The algorithm simply creates a new ordering of all the basic blocks on the function except the entry block and reorders them according to this new ordering.

4.5 CPRNG

MANY of the transformations depend on an entropy source to take random choices. Here, we will be using a CPRNG for this. The CPRNG uses the pad used for encryption by AES in the CTR mode (or what is the same CTR encrypting blocks made of 0s) skipping any remaining bits until the end of the block. This requires a key and an IV, the IV chosen is 0 (as the algorithm will still be safe even if the IV is known) but the key is generated with the following steps:

1. Extract the entropy from the obfuscation key string by applying AES in CMAC mode using a known key in hexadecimal. The key used is “ABADCEBADABEBEFABADAA-CABACABECEA” (which is the Spanish phrase “Abad, cebada bebe, fabada acaba, cabecea” which would translate as follows: The abbot drinks barley (referring to beer), ends with the fabada (a Spanish dish made with white beans,sausages and pork served with the water they were boiled in), thus nods (out of sleepiness).
2. Use the tag generated by the previous iteration as the AES in CMAC mode key to encrypt one of the following character strings:
 - 2.1. If the transformation applies to a particular function the string is a byte set to 1 followed by the module name then a 0 byte then the function name, another 0 byte then an string dependent on the pass and a final 0 byte.
 - 2.2. If the transformation applies to the whole module the string is: a byte set to 2 followed by the module name then a 0 byte then an string dependent on the pass and a final 0 byte.
3. The resulting tag of this application of AES in CMAC mode will be the key used by the CPRNG to generate the CTR PAD.

A proof of the security of a CPRNG created in this way will be provided later in this document.

5

Code design considerations

5.1 Coding conventions

THE following conventions apply to all the code which was written for this project although particular modifications of these is required given the nature of the original code on which they are applied, these modifications are explained later.

Code is indented using 4 spaces for each opened brace not yet closed. No new line is inserted between keywords or expressions and opening braces.

Variables and arguments can be named as desired, in general iterators are either given a letter starting from i or defined as i followed by an abbreviation of the class being iterated or it followed similarly. This convention was chosen mainly to reduce the development times of the PoC in spite of the maintenance cost. And will probably be dropped if the code is submitted to upstream.

5.1.1 Transformation specific conventions

Classes, methods and functions follow mostly LLVM's conventions, that is classes use camel case starting by a capital letter and methods and functions use camel case starting by a small letter.

5.1.2 Auxiliar library conventions

Classes, methods and functions are given names in underscore separated characters with case depending on the use of abbreviations or words, classes start by a capital letter, methods and functions don't. This will be probably refactored to adjust to LLVM's conventions in later iterations, although the conventions will be kept on the AES code unless it gets merged into the utility library.

5.2 Design choices

A set of libraries and a framework to implement the code had to be chosen. For AES support a slightly modified version of Brian Gladman's AES library [9] was chosen and LLVM's transformation framework was chosen as the main framework. The following subsections expose the implications of such choices.

5.2.1 AES implementation used

Brian Gladman's AES implementation was used and adapted (by adapting the CTR mode so it will provide only the pad). This implementation is used because of its liberal license and has been referred by amongst others Intel's documentation [35].

5.2.2 LLVM Transformations

LLVM transformations inherit from `ModulePass` [30] and `FunctionPass` [29] and are implemented in anonymous namespaces to prevent pollution (common code was moved to the `Utils.cpp` file and implemented in the `Obf` namespace).

The transformations are declared by using the `RegisterPass` [31] template, also a per module ID (depending on the class) is declared as it will be used later for pass identification.

When possible, the transformation keeps the analysis produced and informs of this to the pass manager.

5.2.2.1 Transformation parameters

Parameters are passed by the command line and parsed through the `cl` [16] API in LLVM. A specific parser for probabilities was written. Probabilities are defined as numerator/denominator for example a probability of 50% (1 in 2) would be expressed as $1/2$.

5.2.2.2 Transformation implementation

The obfuscation transformations depend on an obfuscation key being present to work, as such, the presence of this key is used to decide whether the obfuscation transformations should be applied to a particular function or module or not.

6

Transformation implementations

6.1 Obfuscation key

THESE transformation provide handling of the obfuscation keys used by the other transformations. Some of the transformations require a module key which can only be provided in this way, whilst other require function keys which can be forced on all functions with this transformations.

6.1.1 addmodulekey

The addmodulekey transformation simply attaches the specified obfuscation key as named metadata on the module for future use by other transformations.

It is useful as it is the only way to express a module obfuscation key nowadays.

The key can be defined by using the modulekey parameter followed by the string used as module key.

6.1.2 propagatemodulekey

This transformation propagates the module obfuscation key to all the functions in the current module. It will overwrite any keys already set up.

This is useful for testing, applying transformations automatically in some cases, and as an all or none switch.

6.2 Obfuscation

THESE transformations, pick the original code and return a new code which is harder to read by humans whilst being functionally equivalent to the original one, they are mostly based on the ideas provided at [32].

6.2.1 flattencontrol

This transformation applies the control flattening algorithm but it is quite complex given the way in which the LLVM language is implemented.

Also the current implementation could benefit from more modularization of the code. This wasn't done given the time constraints for the project.

The transformation works as follows:

1. Runs the Unify Function Exit Nodes transformation to ensure all exit points on the function are appropriately merged.
2. Ensures the entry block is made of only an unconditional branch instruction
3. Generates a vector containing all the current blocks of the function, these are the ones that will be processed by the algorithm
4. Generates an unreachable block for later use if not already available
5. Creates the `main_node`
6. Runs the phi generation transformation:
 - 6.1. For each instruction on each block being processed:
 - 6.1.1. For each user of an instruction which is an instruction on each BB being processed:
 - 6.1.1.1. If it is a phi node whose use of the instruction refers to a different BB then queue it
 - 6.1.1.2. If the user is on a different BB block queue it
 - 6.1.1.3. If the user is a terminator instruction which can't be processed (right now only branch instructions are processed) then queue it
 - 6.1.2. If there are queued users: create a phi in the main node that will be undefined if coming from the entry point, assign the value when coming from the current block and either keep its value or be undefined when coming from other blocks depending on the keep value flag
 - 6.1.3. For each queued instruction:

- 6.1.3.1. Replace its uses of the instruction for the new phi node
- 6.1.4. Clear the queue
7. Runs the phi moving algorithm:
 - 7.1. For each phi on each BB being processed:
 - 7.1.1. Generate a new phi copying the value used by the phi for each input block it contains, mark this blocks as parsed
 - 7.1.2. For every block not marked as parsed:
 - 7.1.2.1. If it is the entry block mark the value as undefined
 - 7.1.2.2. Otherwise mark the value as the previous value of the phi
 - 7.1.3. Replace the old phi by the new one.
 8. Runs the unknown terminator split algorithm:
 - 8.1. For each block with a terminator instruction which can't be parse
 - 8.2. Split the block before the terminator instruction so it will contain instead a branch
 9. Run the basic block identifier generation algorithm
 - 9.1. For each terminator instruction in each block
 - 9.1.1. Mark the destination in the set
 - 9.2. Convert the set into a vector
 - 9.3. Randomize the order of the elements of the vector
 - 9.4. Map each block with the value referring to it based on the new order
 10. Run the terminator instruction replacement algorithm:
 - 10.1. For each basic block to be processed:
 - 10.1.1. If the instruction is a conditional branch:
 - 10.1.1.1. Add a select instruction before choosing the value of the next successor depending on the condition of the branch
 - 10.1.1.2. Use the select as the main phi entry for this block
 - 10.1.2. Otherwise: use the block identifier of the destination block for the main phi entry
 - 10.1.3. Replace the terminator by an unconditional branch to the main phi
 11. Terminate the main block with a switch instruction depending on the value of the main phi mapping each destination block identifier to the block.

6.2.2 obfuscateconstants

This transformation applies the constant obfuscation algorithm.

The current implementation could separate the different transformations in their own modules for simplicity but this was not done to speed up the development of the PoC.

Also, the move to an array method could add random data when expanding the constants to make inferring the size harder or use a single byte constant so bigger constants would be divided in smaller ones and then reassembled. Finally, randomly reordering the array would make the resulting array impossible to read.

The transformation works as follows:

1. For each Module:
 - 1.1. Generate a pointer to an array containing the constants moved to memory
 - 1.2. For each function:
 - 1.2.1. If no obfuscation key found go to the next function
 - 1.2.2. Generate the corresponding PRNG instance
 - 1.2.3. For each basic block:
 - 1.2.3.1. For each operand on each phi
 - 1.2.3.1.1. Run the obfuscate use function
 - 1.2.3.1.2. For each operand which can be obfuscated¹ on each non phi instruction:
 - 1.2.3.2.1. Run the obfuscate use function
 - 1.2.3.2. For each operand which can be obfuscated¹ on each non phi instruction:
 - 1.2.3.2.1. Run the obfuscate use function
 - 1.3. Generate the array with the values of the moved constants
 - 1.4. Make the pointer point to the array

The obfuscate use function takes an use and works as follows:

1. If the use is of a constant:
 - 1.1. Call the obfuscate constant function for that constant
 - 1.2. If the function returns a new value:
 - 1.2.1. Increase the obfuscated uses constant
 - 1.2.2. Replace the operand of the use by the new value

¹Some LLVM instructions and some calls to intrinsics have operands which must be a constant, for example the alignment in a load instruction or the destinations on a switch instructions. These constants cannot be replaced by code returning said constants.

Finally, the obfuscate constant function takes a constant and works as follows:

1. If the constant is an integer:
 - 1.1. If the length of the constant is smaller or equal to the size of the constants which can be moved to the array and the result of two sided dice throw is 1:
 - 1.1.1. Create a new integer constant with the place where the constant will be placed on the array
 - 1.1.2. Push back the old constant to the array
 - 1.1.3. If the integer constant is chosen to be reobfuscated: increase the counter and call obfuscate constant for it
 - 1.1.4. Create a load instruction to load the address of the constant array
 - 1.1.5. Create a pointer to the new constant from the previously loaded address
 - 1.1.6. Load the data from that pointer
 - 1.1.7. If the original constant is smaller than the array size truncate it.
 - 1.2. Otherwise:
 - 1.2.1. Generate a random constant integer fc
 - 1.2.2. If fc should be reobfuscated (with half probability):
 - 1.2.2.1. increase the reobfuscation counter
 - 1.2.2.2. call obfuscate constants for fc
 - 1.2.3. Choose one of the three options randomly:
 - 1.2.3.1. Option 1:
 - 1.2.3.1.1. Set the second constant (sc) to the original value minus fc
 - 1.2.3.1.2. Set the operation to add
 - 1.2.3.2. Option 2:
 - 1.2.3.2.1. Set the second constant (sc) to the original value plus fc
 - 1.2.3.2.2. Set the operation to subtract
 - 1.2.3.3. Option 3:
 - 1.2.3.3.1. Set the second constant (sc) to the original value xor fc
 - 1.2.3.3.2. Set the operation to xor
 - 1.2.4. If sc should be reobfuscated (with half probability):
 - 1.2.4.1. increase the reobfuscation counter

1.2.4.2. call obfuscate constants for sc

1.2.5. Add a instruction applying operation to sc and fc

2. Return the new value

6.3 Polymorphic

THE polymorphic transformations don't aim at making the code be harder to read but at making it different every time they are run according to the results of a PRNG. This, results in smaller penalties for their usage but can make the code easier to reverse.

6.3.1 bbsplit

This transformation will go over all the basic blocks of the function and for each block decide on splitting the block for each instruction except the phi nodes and the first non phy instruction.

As splitting can mess up with the blocks list all the initial BBs are stored on a vector and then iterate over this vector.

The probability of splitting a block at each particular point can be adjusted by using the splitprobability parameter, keep in mind that setting it to one will result in each instruction being split.

6.3.2 randbb

This transformation applies the reordering of basic blocks algorithm to each function.

The transformation works as follows:

1. For each function:
 - 1.1. If no obfuscation key is found return
 - 1.2. Generate a list of all the basic blocks in the function except the entry block
 - 1.3. Reorder the list randomly
 - 1.4. Make the insertion point be the entry point block
 - 1.5. For each basic block from the list:
 - 1.5.1. If the insertion point is different from the extracted block:
 - 1.5.1.1. Move the block after the insertion point
 - 1.5.1.2. Make the new block be the new insertion point

6.3.3 randins

This transformation applies the reordering of instructions algorithm to each basic block.

The code could have benefited from separating the phi node handling function from the more complex handling of normal instructions. Again, this wasn't done because of the time constraints.

The transformation works as follows:

1. For each function:
 - 1.1. If no obfuscation key is found return
 - 1.2. For each basic block:
 - 1.2.1. Generate a list of all the phi nodes in the basic block
 - 1.2.2. Reorder the list randomly
 - 1.2.3. Make the insertion point be the first phi node in the basic block
 - 1.2.4. For each phi node from the list:
 - 1.2.4.1. If the insertion point is different from the extracted phi node:
 - 1.2.4.1.1. Move the phi node before the insertion point
 - 1.2.4.2. Otherwise:
 - 1.2.4.2.1. Point the insertion point to the next phi node
 - 1.2.5. For each normal instruction I:
 - 1.2.5.1. If the instruction may have side effects:
 - 1.2.5.1.1. for each instruction in the basic block J after I:
 - 1.2.5.1.1.1. if J is a memory read or has side effects
 - 1.2.5.1.1.1.1. Add I to the dependency list for J in the map
 - 1.2.5.2. If the instruction is a memory read:
 - 1.2.5.2.1. for each instruction in the basic block J after I:
 - 1.2.5.2.1.1. if J has side effects
 - 1.2.5.2.1.1.1. Add I to the dependency list for J in the map
 - 1.2.5.3. For each operand of I, J:
 - 1.2.5.3.1. If J isn't an instruction: go to the next J
 - 1.2.5.3.2. If J is a phi node: go to the next J

- 1.2.5.3.3. If J is on a different block from I: go to the next J
- 1.2.5.3.4. Insert J into the dependency list for I
- 1.2.5.4. If I has no dependencies push it into the candidate list
- 1.2.6. Set the insertion point as the first instruction after the phi nodes
- 1.2.7. While the candidate list is not empty:
 - 1.2.7.1. Pick a random instruction, I from the list
 - 1.2.7.2. If the insertion point is different from I:
 - 1.2.7.2.1. Move I before the insertion point
 - 1.2.7.3. Otherwise:
 - 1.2.7.3.1. Point the insertion point to the next instruction
 - 1.2.7.4. For each use of I, J:
 - 1.2.7.4.1. If J is an instruction and is in the dependency map:
 - 1.2.7.4.1.1. Remove I from the dependency list of J
 - 1.2.7.4.1.2. If the dependency list of J is empty:
 - 1.2.7.4.1.2.1. Remove the dependency list from the map
 - 1.2.7.4.1.2.2. Add J to the candidate list

6.3.4 randfun

This transformation applies the reordering of functions algorithm to each module.

Although not interesting to binary patch obfuscation this was developed for the interest it provided in hardening of code at compilation time.

The transformation works as follows:

- 1. For each module:
 - 1.1. If no obfuscation key is found return
 - 1.2. Generate a list of all the functions in the module
 - 1.3. Reorder the list randomly
 - 1.4. Make the insertion point be the first function in the module
 - 1.5. For each function from the list:
 - 1.5.1. If the insertion point is different from the extracted function:

1.5.1.1. Move the function after the insertion point

1.5.1.2. Make the new function be the new insertion point

6.3.5 randglb

This transformation applies the reordering of globals algorithm to each module.

Although not interesting to binary patch obfuscation this was developed for the interest it provided in hardening of code at compilation time.

The transformation works as follows:

1. For each module:

1.1. If no obfuscation key is found return

1.2. Generate a list of all the globals in the module

1.3. Reorder the list randomly

1.4. Make the insertion point be the first global in the module

1.5. For each global from the list:

1.5.1. If the insertion point is different from the extracted global:

1.5.1.1. Move the global after the insertion point

1.5.1.2. Make the new function be the new insertion point

6.3.6 swapops

This transformation applies the reordering of operands algorithm to each module, this usually results in different registers being allocated on the resulting assembly code.

The transformation works as follows:

1. For each function:

1.1. If no obfuscation key is found return

1.2. For each instruction I in each basic block:

1.2.1. If I is not commutative: go to the next I

1.2.2. If I is not a binary operator: go to the next I

1.2.3. If a 1 is rolled in a 2 sided dice:

1.2.3.1. Swap the operands

1.2.3.2. Increment the swapped operands counter

7

Evaluation

7.1 Proof: the CPRNG is a good PRNG

As the CTR mode based CPRNG being used is in the heart of this project's transformations it's important to prove that it follows the properties desirable for any Pseudo Random Number Generation to prove that the use of such generator is adequate.

In the following subsections proof is provided that the CPRNG has the properties of Determinism, Uniformity and Independence and its period is also calculated.

7.1.1 Determinism

Determinism is given by the fact that no random data is used to generate the key used by CTR mode and the original IV is the same, thus as block ciphers need to be deterministic in order to allow decryption on the other side; the CPRNG is deterministic.

7.1.2 Uniformity

Given that block ciphers are a one to one mapping of n-bit blocks to n-bit blocks and the IV is incremented by one each time, the PRNG will cover all the 2^n possible inputs and thus the 2^n possible outputs thus generating the largest possible uniform output.

7.1.3 Independence

Since the mapping done by the encryption algorithm is based on the key used all outputs are independent from each other if the encryption algorithm is a pseudorandom permutation.

7.1.4 Function period

As the counter iterates over a total of the 2^n states which are possible with its n -bits and each input block is mapped to a different and unique output block; the period of the CPRNG is exactly 2^n which should be large enough for any practical use.

7.2 Proof: the CPRNG is secure

IN a similar way, as the CTR mode based CPRNG used can be attacked to infer the decisions taken on the transformations, it's important to prove that it follows the properties desirable for any Cryptographic Pseudo Random Number Generation to prove that the use of such generator is adequate.

In the following subsections proof is provided that the CPRNG has the properties of the resistance to next bit tests provided, the impossibility of deriving the function result if the state is known, and based on this, the resistance to the extension of state compromise extension.

7.2.1 Next bit-test resistance

As long as the block cipher used for the CTR mode is resistant to cryptanalysis it will be impossible to derive the key used and thus the state to predict the next block that will be generated. Inside blocks this property is held as the cipher is a pseudorandom permutation and thus no bit presents a visible dependence from the previous one.

7.2.2 Impossibility to know the result if only the state is known

One of the problems with the CPRNG is that the seed used for the state (the IV of the CTR mode) is known (and is 0). Since the attacker has no way of knowing the key (if the obfuscation key is kept secret) it is impossible for him to know which of all the possible blocks will be generated by AES.

7.2.3 State compromise extension resistance

Since the key used in CTR is hidden and not part of the state (which is only the IV), knowing the value of the IV provides no information as long as the cipher is resistant to known plaintext attacks. As a result, given the impossibility to know the result if only the state is known, even if the state is known and previous and future states can be derived it is impossible for the attacker to know the result of the function thus making the algorithm secure.

7.3 Proof: the key derivation is secure

THE first CMAC iteration is done to use a symmetric cipher as a hash function to summarize the entropy of the obfuscation key string, since CMAC uses the previous AES outputs to calculate the next ones this effectively results in all the entropy from the original key being kept in the resulting tag.

The second CMAC iteration uses the resulting key as the key to encrypt a string made of publicly known data (an identifier depending on the function name being available or not, the module name and the transformation name).

Since the obfuscation key is only used as the key of the CMAC algorithm it is impossible for the attacker to derive it without actually breaking AES. Also the entropy provided by the key and the input string is effectively summarized by CMAC into a smaller string which can be used as key for CTR.

7.4 Reversing the transformations

DURING the evaluation of the implemented transformations it has been discovered that it is possible to reverse all of them. The following sections describe a method for doing so although said method wasn't implemented given the time constraints.

The objective isn't getting back to the original code (doing so is most likely impossible without breaking the CPRNG) but getting a set of transformations that when applied to the original and the obfuscated assembly will result in the same LLVM IR (and thus if the obfuscated code is equal to the one provided previously it will result in the same IR code) and that when the obfuscated assembly is different from the original one, the resulting IR code will be only different in the things that were changed allowing the attacker to focus only on the vulnerability.

The possibility of reversing the transformations, though, depends on the possibility of transforming the resulting assembly back into LLVM's IR. A way to do that would be by modeling each instruction of the assembly language into one or more equivalent instructions in LLVM's IR and transforming register accesses into memory reads and writes (which could be optimized later).

As of now no library able to do that is known, but it is reasonable to think that they may be developed in the future.

7.4.1 Defining a global ordering of values

Values can be constants or instruction results. Defining this ordering matters as it allows to define how to order the "contents" of an instruction, that is their operands:

The value ordering is defined as follows given the instructions I and J:

1. If I and J are constants then order after the constant value
2. If J isn't a constant and I is then I goes before J
3. If I isn't a constant and J is then J goes before I
4. If I and J are instructions then order after the way in which they would be ordered according to the global ordering.

7.4.2 Defining a global ordering of instructions

This is the core of reversing most of the transformations, as when instructions can be ordered then an ordering can also be created for the contents of basic blocks and functions.

The instruction ordering is defined as follows given the instructions I and J in the same basic block:

1. If I depends on J or J depends on I:
 - 1.1. If J is originally before I then I goes after J
 - 1.2. Otherwise J goes after I
2. Otherwise if J and I are of different types of instruction then order according to the instruction type (first by operand count then by an arbitrary order)
3. Otherwise order I and J according to the ordering of their operands.

7.4.3 Reversing the randfun and randglb transformations

These can be reversed quite trivially by reordering globals and functions alphabetically or, when anonymous or with the same name, by their contents values.

7.4.4 Reversing the swapops transformation

This can be reversed once an ordering for values is defined by ordering the operands of the instruction accordingly if the instruction is a candidate for operand swapping.

7.4.5 Reversing the randins transformation

This transformation can be reversed by ordering the instructions according to the global ordering in the basic block. At times two instructions with exactly the same contents may be found. If this happens the instructions may be merged resolving the conflict.

7.4.6 Reversing the obfuscateconstants transformation

To reverse this transformation the only thing that needs to be done is to pass a constant calculation transformation which will replace the instructions by the constant values they calculate and finally remove any unused global variables.

7.4.7 Reversing the flattencontrol transformation

To reverse this one find the phi node that chooses the destination of the main block switch depending on the block which jumped to us, then replace the unconditional jumps by the node chosen on the main block, or when using a select by a conditional block

depending on the select condition and the node that will be chosen in the main block in each side.

Afterwards move the phi nodes to the first basic block where they are used according to the CFG and a liveness analysis and delete the main block.

Finally apply the Unify Function Exit Nodes transformation to ensure both flow graphs are equal.

7.4.8 Reversing the bbsplit transformation

To reverse this one, simply merge any two BBs BB1 and BB2 where BB1 has an unconditional jump to BB2 and BB2 has only BB1 as predecessor.

7.4.9 Reversing the randbb transformation

This transformation can be reversed by ordering the basic blocks by traversing the CFG using breadth first search choosing the basic blocks when two or more are available at the same level according to the order in which their parents were chosen or, when the same parents are there, the contents of the BB itself. If the contents are the same then the blocks may be merged instead.

Since the contents may be different when going back this ordering may not result in the same code on both sides when the BB contents are different. An optimization pass can be used though to reduce these differences.

7.5 Binary patch obfuscation technique evaluation

THE proposed technique consists on obfuscating some of the functions of the code being patched along with the patched function, choosing these extra functions at random.

It is easy to see that if the attacker has no knowledge of which function was modified and can't reverse the transformations he will need to analyze a mean of half of the added functions before finding the changed functions and all of them before he can be certain no other functions are unmodified.

Also, if only a function introduces the security fix then the probability of the reverse engineer finding it after x attempts is inversely proportional to the number of added functions and directly proportional to the number of attempts.

Sadly, an experiment to check how efficient the used obfuscation techniques are couldn't be run but they should be as efficient as the original ones they are based on. This lack of an experiment made it impossible to measure the amount of extra time that is required to analyze obfuscated functions.

8

Conclusions

WE have developed a set of transformations allowing the focused obfuscation of functions so only these will be different on the resulting patch. Said transformations have also been implemented into LLVM.

On the evaluation a proof was provided that given enough interest the proposed polymorphic and obfuscation transformations can be reversed or, when that's not possible, a similar transformation can be applied to both codes to get a minimal set of differences between the original and the modified code. A proof that if the passes can't be reversed the difficulty of finding the security fix increases proportionally to the number of extra obfuscated functions is provided also.

The results of the evaluation can be considered an example of the never ending war between researchers trying to elaborate better obfuscation techniques and attackers trying to reverse them, this situation will end either when a technique which can't be reversed is developed or when newer techniques can't be created by the other side. Sadly, the current situation hints that the second may happen more likely than the first as the ways in which programs can be obfuscated are limited and human thinking can adapt to read obfuscated code.

9

Future development

GIVEN the time constraints many things couldn't be covered on this project. The first thing that should be done in the future is improving the code quality of the transformations so it can be pushed into upstream LLVM.

On the constant obfuscation transformation, improvement can be done on the constant memory fetch obfuscation by using Wang's aliasing method and randomly reordering the constant array.

The Register Swap could be done instead over the resulting assembly by remapping registers (which sadly is API dependent).

Also, an study of the efficiency of the transformations should be run, although some preliminary tests hint at around 6x slowdown.

Finally, other obfuscation techniques could be applied to make the resulting patches harder to analyze by attackers.

Also, as part of the development of this project and discovery that some obfuscation techniques can be used to harden the resulting binaries against certain attacks with apparently negligible impact was made, a more deep research on this topic will be done in the near future.

Bibliography

- [1] Arini Balakrishnan and Chloe Schulze. *Code Obfuscation Literature Survey*. 2005. URL: <http://pages.cs.wisc.edu/~arinib/writeup.pdf>.
- [2] Boaz Barak et al. “On the (im)possibility of obfuscating programs”. In: *Lecture Notes in Computer Science*. Springer-Verlag, 2001, pp. 1–18. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.8717&rep=rep1&type=pdf>.
- [3] Eli Bendersky. *Life of an instruction in LLVM*. Nov. 2012. URL: <http://eli.thegreenplace.net/2012/11/24/life-of-an-instruction-in-llvm/>.
- [4] Jean-Marie Borello and Ludovic Mé. “Code obfuscation techniques for metamorphic viruses”. English. In: *Journal in Computer Virology* 4.3 (2008), pp. 211–220. ISSN: 1772-9890. DOI: [10.1007/s11416-008-0084-2](https://doi.org/10.1007/s11416-008-0084-2). URL: <http://dx.doi.org/10.1007/s11416-008-0084-2>.
- [5] Frederick P. Brooks Jr. “No Silver Bullet Essence and Accidents of Software Engineering”. In: *Computer* 20.4 (Apr. 1987), pp. 10–19. ISSN: 0018-9162. DOI: [10.1109/MC.1987.1663532](https://doi.org/10.1109/MC.1987.1663532). URL: <http://dx.doi.org/10.1109/MC.1987.1663532>.
- [6] Chih-Fan Chen et al. *CONFUSE: LLVM-based Code Obfuscation*. May 2013. URL: http://www.cs.columbia.edu/~aho/cs4115_Spring-2013/lectures/13-05-16_Team11_Confuse_Paper.pdf.
- [7] S. Garg et al. “Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits”. In: *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. Oct. 2013, pp. 40–49. DOI: [10.1109/FOCS.2013.13](https://doi.org/10.1109/FOCS.2013.13). URL: <http://eprint.iacr.org/2013/451.pdf>.
- [8] Anne Gille-Genest. *Pseudo-Random Numbers Generators*. Mar. 2012. URL: https://quanto.inria.fr/pdf_html/mc_random_doc/#x1-40001.2.
- [9] Brian Gladman. *AES and Combined Encryption/Authentication Modes*. 2014. URL: <http://gladman.plushost.co.uk/oldsite/AES/index.php>.

-
- [10] Sebastian Hack and Gerhard Goos. “Optimal Register Allocation for SSA-form Programs in Polynomial Time”. In: *Inf. Process. Lett.* 98.4 (May 2006), pp. 150–155. ISSN: 0020-0190. DOI: [10.1016/j.ipl.2006.01.008](https://doi.org/10.1016/j.ipl.2006.01.008). URL: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4E2D4510A92E0FD0AB2775F446362B18?doi=10.1.1.204.2844&rep=rep1&type=pdf>.
 - [11] Francisco Blas Izquierdo Riera. “Pint, herramienta de simulación basada en trazas Pin”. MA thesis. Polytechnical University of Valencia, Dec. 2012. URL: <http://hdl.handle.net/10251/18304>.
 - [12] Francisco Blas Izquierdo Riera. *The SIV mode of operation result in data leakage with small messages (\leq blocksize) when the authentication part of the key is discovered and how to get data from CMAC*. June 2011. URL: <http://seclists.org/fulldisclosure/2011/Jun/382>.
 - [13] Pascal Junod. *Obfuscator-LLVM*. 2013. URL: <https://github.com/obfuscator-llvm/obfuscator/wiki>.
 - [14] Pascal Junod. *Obfuscator reloaded*. Nov. 2012. URL: http://crypto.junod.info/asfws12_talk.pdf.
 - [15] Cullen Linn and Saumya Debray. “Obfuscation of Executable Code to Improve Resistance to Static Disassembly”. In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS ’03. Washington D.C., USA: ACM, 2003, pp. 290–299. ISBN: 1-58113-738-9. DOI: [10.1145/948109.948149](https://doi.org/10.1145/948109.948149). URL: <http://doi.acm.org/10.1145/948109.948149>.
 - [16] LLVM Project. *CommandLine 2.0 Library Manual*. Mar. 2014. URL: <http://llvm.org/docs/CommandLine.html>.
 - [17] LLVM Project. *LLVM Language Reference Manual*. 2014. URL: <http://llvm.org/docs/LangRef.html>.
 - [18] LLVM Project. *Writing an LLVM Pass*. 2014. URL: <http://llvm.org/docs/WritingAnLLVMPass.html>.
 - [19] Matias Madou, Ludo Van Put, and Koen De Bosschere. “LOCO: An Interactive Code (De)Obfuscation Tool”. In: *Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*. PEPM ’06. Charleston, South Carolina: ACM, 2006, pp. 140–144. ISBN: 1-59593-196-1. DOI: [10.1145/1111542.1111566](https://doi.org/10.1145/1111542.1111566). URL: <http://doi.acm.org/10.1145/1111542.1111566>.
 - [20] Matias Madou et al. *Code (De)Obfuscation*. June 2005. URL: http://escher.elis.ugent.be/publ/Edocs/DOC/P105_076.pdf.
 - [21] Matias Madou et al. “On the Effectiveness of Source Code Transformations for Binary Obfuscation”. In: *Proc. of the International Conference on Software Engineering Research and Practice (SERP06)*. 2006. URL: <https://biblio.ugent.be/publication/374659/file/496495.pdf>.

- [22] Jeongwook Oh. “Fight Against 1-day Exploits: Diffing Binaries vs Anti-diffing Binaries”. In: *Black Hat USA 2009 //Media Archives*. Las Vegas, NE, USA, 2009, p. 1. URL: <http://www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinaries-PAPER.pdf>.
- [23] Axel ”0vercl0k” Souchet. *Obfuscation of steel: meet my Kryptonite*. July 2013. URL: http://download.tuxfamily.org/overcloblog/Obfuscation%20of%20steel%3a%20meet%20my%20Kryptonite/Overcl0k_Obfuscation_of_steel_meet_kryptonite.pdf.
- [24] StatCounter. *Top 8 Operating Systems from Feb 2013 to Jan 2014*. Feb. 2014. URL: <http://gs.statcounter.com/#all-os-ww-monthly-201302-201401>.
- [25] StatCounter. *Top 9 Browsers from Feb 2013 to Jan 2014*. Feb. 2014. URL: <http://gs.statcounter.com/#all-browser-ww-monthly-201302-201401>.
- [26] Teja Tamboli. “Metamorphic Code Generation from LLVM IR Bytecode”. MA thesis. San José State University, 2013. URL: http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1305&context=etd_projects.
- [27] The Clang Team. “Clang” *CFE Internals Manual*. 2014. URL: <http://clang.llvm.org/docs/InternalsManual.html#how-to-add-an-attribute>.
- [28] S.K. Udupa, S.K. Debray, and M. Madou. “Deobfuscation: reverse engineering obfuscated code”. In: *Reverse Engineering, 12th Working Conference on*. Nov. 2005, 10 pp.–. DOI: 10.1109/WCRE.2005.13. URL: <http://dx.doi.org/10.1109/WCRE.2005.13>.
- [29] University of Illinois at Urbana-Champaign. *LLVM API Documentation: llvm::FunctionPass Class Reference*. Mar. 2014. URL: http://llvm.org/docs/doxygen/html/classllvm_1_1FunctionPass.html.
- [30] University of Illinois at Urbana-Champaign. *LLVM API Documentation: llvm::ModulePass Class Reference*. Mar. 2014. URL: http://llvm.org/docs/doxygen/html/classllvm_1_1ModulePass.html.
- [31] University of Illinois at Urbana-Champaign. *LLVM API Documentation: llvm::RegisterPass< passName > Struct Template Reference*. Mar. 2014. URL: http://llvm.org/docs/doxygen/html/structllvm_1_1RegisterPass.html.
- [32] Chenxi Wang. “A Security Architecture for Survivability Mechanisms”. PhD thesis. Charlottesville, VA, USA: Faculty of the School of Engineering and Applied Science at the University of Virginia, 2001. ISBN: 0-493-08929-2. URL: <http://www.cs.virginia.edu/~jck/publications/wangthesis.pdf>.
- [33] Gregory Wroblewski. “General Method of Program Code Obfuscation”. PhD thesis. Wrocław University of Technology, 2002. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.9052&rep=rep1&type=pdf>.

- [34] Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey”. In: *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*. Nov. 2010, pp. 297–300. DOI: [10.1109/BWCCA.2010.85](https://doi.org/10.1109/BWCCA.2010.85). URL: <http://dx.doi.org/10.1109/BWCCA.2010.85>.
- [35] Dan Zimmerman. *Got AES Performance?* Oct. 2010. URL: <http://software.intel.com/en-us/blogs/2010/10/14/got-aes-performance>.

A

Using the tools

THE obfuscation transformations are compiled into their own library at `lib/Obf.so` and need to be loaded explicitly when using `opt` by using the `--load` switch. This is done mainly to keep the code isolated from the rest of the tools making it easier to integrate.

The library can be loaded doing, for example this:

Listing A.1: Loading the obfuscation library

```
1 $ opt --load ./Release+Asserts/lib/Obf.so
```

`opt` takes as input an LLVM IR program and outputs another, transformed, one. The transformations are applied in the order given. The following switches will add a pass by the different transformations:

- `-addmodulekey` Enables the transformation for adding a module key.
- `-bbsplit` Enables the `bbsplit` transformation which randomly splits basic blocks
- `-flattencontrol` Enables the control flattening transformation which will flatten the marked functions
- `-obfuscateconstants` Enables the constant obfuscation transformation
- `-propagatmodulekey` Enables the module key propagation to the module functions
- `-randbb` Enables the randomly reordering of BBs transformation
- `-randfun` Enables the transformation for randomly reordering functions
- `-randglb` Enables the transformation for randomly reordering globals

-randins Enables the transformation doing the dependence based random reordering of instructions

-swapops Enables the transformation for randomly swapping instruction operands

Aside, some of the transformations have a set of tunable parameters, which can be modified by using the following flags:

-modulekey *string* Defines the key inserted in the module by **-addmodulekey**

-splitprobability *probability* Specifies the probability of splitting the block at each instruction for **bbsplit**

-reobfuscationprobability *probability* Specifies the probability of reobfuscating a constant

Also the order in which this transformations are run can affect the resulting code and the effectiveness of the transformations thus the following order is recommended:

1. Any optimization transformations
2. The **-addmodulekey** transformation
3. The **-propagatmodulekey** transformation
4. The **-bbsplit** transformation
5. The **-flattencontrol** transformation
6. The **-obfuscateconstants** transformation
7. The **-randins** **-randbb** **-randfun** **-randglb** and **-swapops** transformations

An example of a complete call to **opt** would be the following:

Listing A.2: Using **opt** to obfuscate LLVM code

```
$ opt --load ./Release+Asserts/lib/Obf.so -addmodulekey
  -modulekey "Example_key" -propagatmodulekey
  -bbsplit -splitprobability 1/8 -flattencontrol
  -obfuscateconstants -reobfuscationprobability 1/5
  -randins -randbb -randfun -randglb -swapops <
  input.bc > output.bc
```

In order to work with **clang** the **-emit-llvm** option must be added and an extra call to **clang** for linking has to be added. The following line which is explained later contains the procedure to compile and link a c file:

Listing A.3: Compiling an obfuscated binary using **clang** and **opt**

```
$ clang file.c -O3 -c -emit-llvm -o - | opt --load
  ./Release+Asserts/lib/Obf.so -addmodulekey
  -modulekey "Example_key" -propagatmodulekey
```

```
-bbsplit -splitprobability 1/8 -flattencontrol  
-obfuscateconstants -reobfuscationprobability 1/5  
-randins -randbb -randfun -randglb -swapops | clang  
-x ir -
```

Here a pipeline from `clang` to `opt` and then back to `clang` is generated.

The first call to `clang` compiles the provided C file, runs the level 3 standard optimizations with `-O3` and stops before linking with `-c` it then emits the LLVM bytecode with `-emit-llvm` and outputs it to the standard output with `-o -`

The call to `opt` parses the generated bytecode as explained before.

Finally the last call to `clang` uses `-x ir` to specify the input contains LLVM's IR (in bytecode form) and a single `-` to make `clang` take the input from the standard input. This will link and compile the IR code generated by `opt` into the `a.out` file. If object code is desired then the `-c` option can be used, in a similar way, to specify the desired output file, the `-o` option can be used.

B

Popularization

B.1 Programming computers

LIKE many other digital systems, computers work in binary, that is a language where two clearly different values exist. These values are usually referred as 0 and 1 and each one of them is considered a bit.

As these values by themselves would allow for only two possible states they can be grouped to provide a larger array of possible values, for example if 2 values were grouped the following 4 states could then be obtained: 00, 01, 10 and 11. In a similar way with 3 values you can get 8 different states and in general with n values you can get 2^n states.

By themselves, these groups of values set to 0 or 1 mean nothing, but it is possible to use them to encode things like the colors of an image or the amount of money that you have in your bank account. This is done by providing a meaning to each of the bits in the group so each of the two possible values will affect the meaning of the group in one way or another.

You may also be aware that computers are programmable, this means that they use some instructions to know what should they do with the groups of bits they work with. These instructions are also encoded using groups of ones and zeros so the computer knows for example that it needs to get a value from a particular place or add the values it can find on two places and put the result on a third place. Also, the meaning of these bit sequences is heavily dependent on the computer type as different computer designs assign different meanings.

B.1.1 Assembly

When humans want to give orders or transmit information to each other they don't say "01110010101" but use words like "bring me water". As a result of this, it's hard for humans to understand or speak directly in binary.

To fix this problem, assembly languages were created, an assembly language is a compromise between the high level languages used by humans and the binary languages used by machines. For example, on an IntelTM processor (like the one on most desktops) the code "0000000011000011" means "add the value of the bits on the locations al and bl and store the result in the location al" which would be written in assembly as "add %al,%bl".

But, as stated, machines have no way of understanding that "add %al,%bl" means actually "0000000011000011" without a special program that can do the translation. This program is called an assembler. In a similar way it is interesting in some situations to have a program telling you that "0000000011000011" means "add %al,%bl" such program is known as disassembler.

Assembly languages usually allow for some small levels of abstraction which simplify the reading for humans, for example they may allow to add comments explaining what different parts of the code do or add descriptive names to different locations where information can be stored for processing. When converting the program into a set of instructions written in binary which the machine can understand, this information is discarded as it is unnecessary for the machine (and in many cases it can't be encoded anyways). As a result, when the disassembler converts back the binary instructions the machine can understand into assembly language, this information will be missing making the code harder to understand.

The main advantage of assembly is that it allows the programmers use a language which is easier to understand by humans whilst still exposing all the capacities of the machine being used. But, in exchange, some time needs to be invested in translating the program into the language the machine understands although this only needs to be done a single time. Also, as mappings can be created both ways, it is possible to convert back the original binary code into an assembler instruction which is easier to read by trained humans.

The main disadvantages is that assembly is still hard to understand for humans (as they need to picture the machine doing the instructions in their heads to see what the code does) and that each machine uses a different assembly language as they have different characteristics to expose.

B.1.2 Programming languages

In order to overcome the disadvantages of assembly languages, programming languages were created.

A programming language is a construct which abstracts the details of the machine allowing the same code to be used on machines with different features and tries to provide an interface which is easier for humans to work with. This is done at the expense of more processing time and more complex programs being needed in order to convert the programs wrote into these languages into equivalent programs in binary which the machine can understand.

There are different paradigms which are differentiated mostly by the way in which they allow the programmer to model the program.

As more abstraction is added, the resources provided by the computer are used less efficiently but the program can be used more easily on other computers. Also, a more simple interface is provided which allows the programmer to model the program in a language nearer to that of the problem he is trying to solve. The same applies in the reverse as more and more details of the machine are provided by the language.

B.2 Compilers

As said before machines work using binary languages which tell them exactly what to do to solve a problem, thus a program able of converting the instructions given in those more abstract programming languages into a program using binary language that the machine can understand is needed. This program is known as a compiler.

In general, compilers don't do this transformation into machine language directly as that would result in very complex programs that would need to do everything at the same time. Instead, they go over a set of stages which convert more abstract languages into either the same language or a less abstract one. This new program will keep the same meaning as the original.

For example, a program compiled using the LLVM suite would first be converted into an assembly like language that hides most of the limitations of real machines, then optimized into a faster program in this same language, then converted into a representation where operands indicate which are they operators, optimized again into a faster program using this representation and rewritten again so the final representation matches the limits imposed by the destination machine. Finally, this representation would be converted into assembly language and passed to an assembler to convert that program into one in binary language the machine can understand.

During the compilation process compilers, in a way similar to assemblers, discard the information that isn't needed by the machine to understand the program. This along with the fact that there is no direct mapping from the machine instructions to the abstract representation used originally makes it harder to get the original program (minus the discarded information) from the binary one provided. Anyways, despite these limitations, some programs are able to recognize patterns on the code generated for different structures by compilers. These kind of programs are called decompilers.

B.3 Antireverse engineering

REVERSE engineering is the process of taking apart the different pieces that make a product in order to understand how this product works. In a similar way, when applied to a programming context, it is the process of analyzing the machine code contained by a program in order to understand what it does and how it does it.

There are many uses of reverse engineering: understanding how a program works, understanding the results it produces to generate equivalent results with your own program or interpret them, modifying the program to override code to enforce license restrictions or even detecting flaws in the program that can be used.

Because of the possibility of others doing reverse engineering, programmers try to make it harder for others to understand the machine code produced after the compilation. There are different ways of doing this.

One way of making programs harder to reverse engineer consists on removing any unnecessary human understandable information which could be contained in the program, this process is generally known as stripping as the program is “stripped” to the minimum necessary to be executed by the machine.

Another way is thwarting the efforts of decompilers or disassemblers by exploiting some properties of the machine code. But neither this nor the previous technique will stop people who can understand machine code and thus understand the program.

A last way is making the resulting machine code harder to read by humans. This is the process known as obfuscation. This process though, usually comes with a price as it may result in larger and slower programs. Also, as shown by [2] some programs can’t be obfuscated.

B.3.1 Obfuscation techniques

As interest in the area grew, more and more techniques to make programs harder to understand were developed. In a similar way, more and more effective methods of reversing programs obfuscated using these techniques were created. This has resulted in an arms race where one group develops stronger techniques and the other stronger methods to override them.

Some of these techniques focus simply on making the program different from the original one. These techniques are called polymorphism techniques as they make the program take different shapes. Others instead try to modify the structure of the program to make the resulting code harder to read.

B.3.1.1 Control flattening

Some instructions tell the computer to continue executing instructions other than the next one. These instructions can be also executed only when certain conditions are met, this allows for the creation of loops which will repeat the same sequence of instructions either forever or until a condition is met.

Control flattening works by replacing all these instructions by a jump to the same sequence of instructions. This sequence will then jump to the originally intended destination based on the information passed before jumping there.

To help understand the results this recipe will be obfuscated:

1. Put 4 eggs on an empty dish
2. Add $\frac{1}{2}$ glass of oil to that dish
3. Add 500 grams flour in an empty bowl
4. Add 1 glass of water to that bowl
5. Add a spoon of yeast to that bowl
6. Add the contents of the dish to that bowl
7. Knead the mix
8. If the mix isn't a consistent dough then repeat step 7
9. If the mix isn't leaved repeat step 9
10. Start the oven with a temperature of 180°
11. If the oven isn't at 180° repeat step 11
12. Put the mix in the oven
13. If the mix isn't baked repeat step 13
14. Remove the mix from the oven
15. Shut down the oven
16. You are done

As it can be seen it just provides some simple steps to cook some food using an oven. Computers would work in a similar way to a human following the steps indicated in that recipe when running a program. Now, if the recipe was obfuscated using control flattening it would look like this:

1. Put 4 eggs on an empty dish
2. Add $\frac{1}{2}$ glass of oil to that dish

3. Add 500 grams flour in an empty bowl
4. Add 1 glass of water to that bowl
5. Add a spoon of yeast to that bowl
6. Add the contents of the dish to that bowl
7. Knead the mix
8. If the mix isn't a consistent dough then write 1 on a paper otherwise write 2
9. Go to step 21
10. If the mix isn't leaved then write 3 on a paper otherwise write 4
11. Go to step 21
12. Start the oven with a temperature of 180°
13. If the oven isn't at 180° then write 5 on a paper otherwise write 6
14. Go to step 21
15. Put the mix in the oven
16. If the mix isn't baked then write 7 on a paper otherwise write 8
17. Go to step 21
18. Remove the mix from the oven
19. Shut down the oven
20. You are done
21. If the paper says 1 go to step 7
22. If the paper says 2 go to step 10
23. If the paper says 3 go to step 10
24. If the paper says 4 go to step 12
25. If the paper says 5 go to step 13
26. If the paper says 6 go to step 15
27. If the paper says 7 go to step 16
28. If the paper says 8 go to step 18

The result of this transformation is that the reverse engineer has a harder time understanding how the flow of instructions will work inside the program, as he will first see that all the jumps go to the same place and from there to the other instructions.

B.3.1.2 Constant obfuscation

Many programs need constant values to work, for example if you want to calculate the price of a product including a fixed amount of taxes you'd need to know which is that amount so you can add it to the original price. The idea behind constant obfuscation is making these values harder to find.

For example imagine that you make a program which will add 4 to the value received as input. You could do it just by adding 4 or by adding the result of $\frac{(2-1+5)\times 2}{3}$. The second one is obviously harder to understand. In a similar way you could add the result of fetching the information from a particular memory address which will return 4.

Using the previous recipe as an example this transformation would look as follows:

1. Write $\frac{1}{2}$ on a paper
2. Put $\frac{(2-1+5)\times 2}{3}$ eggs on an empty dish
3. Add the amount on the paper glass of oil to that dish
4. Add $4000 - 7 \times 500$ grams flour in an empty bowl
5. Add $\frac{8}{2^3}$ glass of water to that bowl
6. Add $\frac{3 \times 12}{24 + 6 \times 2}$ spoon of yeast to that bowl
7. Add the contents of the dish to that bowl
8. Knead the mix
9. If the mix isn't a consistent dough then repeat step 8
10. If the mix isn't leaved repeat step 10
11. Start the oven with a temperature of 180°
12. If If the oven isn't at 180° repeat step 12
13. Put the mix in the oven
14. If the mix isn't baked repeat step 14
15. Remove the mix from the oven
16. Shut down the oven
17. You are done

This technique aims at making values which are the same along the execution of the program harder to read thus making it harder to find these constant points and use them as references to understand how the program works.

B.3.1.3 Register swap

In computers, some places where binary information can be stored have special meanings like being the place where the next instruction to be executed can be found or the color that needs to be put in a particular place of your screen. But the majority of them have no other meaning than the one given by the program being run.

What this technique does is randomly change the meaning for the program of pairs of these otherwise meaningless locations resulting in a different program.

Using the previous recipe as an example, the result would be this:

1. Put 4 eggs on an empty bowl
2. Add $\frac{1}{2}$ glass of oil to that bowl
3. Add 500 grams flour in an empty dish
4. Add 1 glass of water to that dish
5. Add a spoon of yeast to that dish
6. Add the contents of the bowl to that dish
7. Knead the mix
8. If the mix isn't a consistent dough then repeat step 7
9. If the mix isn't leaved repeat step 9
10. Start the oven with a temperature of 180°
11. If If the oven isn't at 180° repeat step 11
12. Put the mix in the oven
13. If the mix isn't baked repeat step 13
14. Remove the mix from the oven
15. Shut down the oven
16. You are done

As you can see the ingredients that would be in the dish were changed with those on the bowl. The change may not seem so big at first but if you check carefully, it made 6 out of the 16 instructions of the program be different.

The result of this technique is code which is different every time it's compiled thus making it harder for a reverse engineer to find the differences.

B.3.1.4 Instruction Reordering

The last of the techniques being applied consists on randomly reordering the instructions a program executes as long as there are no dependencies on them.

Using the same recipe used earlier this transformation would look as follows:

1. Add $\frac{1}{2}$ glass of oil to an empty dish
2. Put 4 eggs on that dish
3. Add 1 glass of water to an empty bowl
4. Add the contents of the dish to that bowl
5. Add a spoon of yeast to that bowl
6. Add 500 grams flour in that bowl
7. Knead the mix
8. If the mix isn't a consistent dough then repeat step 7
9. If the mix isn't leaved repeat step 9
10. Start the oven with a temperature of 180°
11. If If the oven isn't at 180° repeat step 11
12. Put the mix in the oven
13. If the mix isn't baked repeat step 13
14. Shut down the oven
15. Remove the mix from the oven
16. You are done

A similar thing can be done by reordering the sequences of instructions which will always be executed in the same order. The recipe could then be modified to look like this:

1. Go to step 8
2. Remove the mix from the oven
3. Shut down the oven
4. You are done
5. Start the oven with a temperature of 180°
6. If the oven isn't at 180° repeat step 6
7. Go to step 15

8. Put 4 eggs on an empty dish
9. Add $\frac{1}{2}$ glass of oil to that dish
10. Add 500 grams flour in an empty bowl
11. Add 1 glass of water to that bowl
12. Add a spoon of yeast to that bowl
13. Add the contents of the dish to that bowl
14. Go to step 20
15. Put the mix in the oven
16. If the mix isn't baked repeat step 16
17. Go to step 2
18. If the mix isn't leaved repeat step 18
19. Go to step 5
20. Knead the mix
21. If the mix isn't a consistent dough then repeat 20
22. Go to step 18

The result of this technique is a program where the order of the elements is changed every time thus making it harder to find the differences from those made on the original program.

B.3.2 Focused obfuscation

To allow for some balance between the penalties introduced by obfuscation and the benefits it provides by making programs harder to reverse engineer, the obfuscation techniques are focused in only some parts of the program.

This provides some benefits: first, only the obfuscated parts of the program will change (thus you have to send less changes to user when he needs to update the program to a new version). Second, only the obfuscated parts will receive the penalties introduced by the obfuscation techniques (thus reducing the total impact). Finally, by using a secret number to define how the techniques will be applied to different parts of the program (or not applied at all) it is possible to generate the same program always making updating already obfuscated programs easier as the parts using the same number will remain the same.

The main drawback of this technique is that by focusing the obfuscation on particular parts of the program the attacker can concentrate its efforts on said parts as he'll

expect the interesting parts to be there. This problem can be solved by choosing many uninteresting parts to be obfuscated too.

B.3.3 Compiler-level obfuscation

It's possible to create compilers which will obfuscate programs as they process them. Doing so comes with some advantages.

One of them is that this simplifies the process for the user of the compiler as he just needs to tell the compiler to obfuscate the program.

Another one is that as the obfuscation technique can be applied in a machine independent language the obfuscation code can be used across machines using different designs.

A final one is that it simplifies the process of focusing the obfuscation techniques as the user can just mark the structures that need to be obfuscated.

The problem is that some obfuscation techniques rely on features which are not modeled by the language used when the obfuscation is applied, as a result these techniques can't be implemented using that language although they may be implemented in one which is nearer to the one the computer understands.

B.4 Deobfuscation

It's possible to reverse the obfuscation techniques which have been implemented in a more or less automated manner. Despite tools to do this don't exist yet these may be developed in the near future as interest in doing so increases. Because of this, care should be taken into looking for new developments in the field of research before using one technique or another as they may only introduce performance penalties without providing any benefit.

B.4.1 Control unflattening

The idea behind this technique is finding the code block where the real control flow of the program is decided and then moving these decisions to the jumps going to said code block. As the decisions are constant values it's reasonably possible to do this automatically.

B.4.2 Constant deobfuscation

The idea used in this case is that as constants will keep the same value during the whole execution they can be calculated once found, thus returning the original values instead of the obfuscated ones.

B.4.3 Register swap

If you define a way to order the places where information is stored based on the instruction being executed and the dependencies amongst instructions you can then order all the possible places where information can be stored and assign them based on the ordering you defined before. By doing this on both sides you should end with barely similar programs.

B.4.4 Instruction Reordering

By using the previously defined ordering you can also order the instructions in both the original and the modified program thus reducing the amount of changes between them.

B.5 Program updates

DEVELOPERS may have many reasons to create new versions of programs and send them to the users of that program. In some cases they may have added new features to the program, whilst in others they may have fixed errors that have been discovered. In some situations these errors are reasonably harmless, but when they can be used by a third party to make the program behave in an undesired way they are considered security vulnerabilities.

Programmers can just send the updated version of the program to its users but this is generally inefficient as most of the program will remain the same and can even be problematic if the program is large enough. To prevent this, instructions explaining how to get the new version of the program from the older one are sent instead. These instructions are usually known as a patch and they can be applied automatically by a special program.

Using patches comes with some drawbacks, first the new version of the program has to be similar enough to the old version for the patch to be smaller than the final program. For example, if polymorphic techniques are applied the whole program would change making this process inefficient.

Another drawback is that an attacker can focus in the changes introduced by the patch to find if security vulnerabilities were fixed and once found, exploit them on the users who haven't updated the program yet. This kind of attack is known as 1-day exploit as it is done after the updated program is released.

B.6 Practical example

SUPPOSE a developer is reported a security vulnerability in a program he has developed, he fixes the issue and prepares a patch. In order to prevent an attacker from simply looking at the changes introduced in order to fix the program the developer could then obfuscate the part of the program that needed to be changed.

As said previously, the attacker can still focus his efforts on the parts of the program that were changed even if they are obfuscated so the developer then decides to obfuscate and modify also other parts of the program along with the one that got fixed. This way not the whole program gets changed and the attacker now needs to read a mean of half of the changes introduced before he can find the one which fixes the vulnerability.

As can be seen the techniques used don't prevent the attacker from eventually finding the issue being fixed and maybe exploiting it in the computers of those who haven't still upgraded the program, but they can delay the attacker and, by doing so, allow more users to update the program before the attacker can abuse the vulnerability.

But as you probably have inferred from the previous chapters, the use of obfuscation techniques can make the program less efficient. Thus the developer should release an non obfuscated patch after enough time for the users to upgrade the program has passed.

C

Source code

C.1 Patches to LLVM

```
1 --- lib/Transforms/Makefile      (revision 192535)
2 +++ lib/Transforms/Makefile      (working copy)
3 @@ -9,6 +9,6 @@
4
5  LEVEL = ../..
6 -PARALLEL_DIRS = Utils Instrumentation Scalar InstCombine IPO
6   ↪Vectorize Hello ObjCARC
7 +PARALLEL_DIRS = Utils Instrumentation Scalar InstCombine IPO
7   ↪Vectorize Hello ObjCARC Obf
8
9  include $(LEVEL)/Makefile.config
```

C.2 Patches to Clang

```

1  --- tools/clang/include/clang/Basic/Attr.td      (revision 192535)
2  +++ tools/clang/include/clang/Basic/Attr.td      (working copy)
3  @@ -565,6 +565,12 @@
4      let Subjects = [ParmVar];
5  }
6
7  +def ObfKey : InheritableAttr {
8  +  let Spellings = [GNU<"obfkey">, GNU<"obfuscation_key">,
9  +    ↪GNU<"obfuscationkey">, GNU<"ObfuscationKey">];
10 +  let Subjects = [Function];
11 +  let Args = [StringArgument<"Key", 1>];
12 +}
13
14  def ObjCException : InheritableAttr {
15      let Spellings = [GNU<"objc_exception">];
16  }
17
18  --- tools/clang/lib/CodeGen/CodeGenModule.cpp    (revision 192535)
19  +++ tools/clang/lib/CodeGen/CodeGenModule.cpp    (working copy)
20  @@ -626,6 +626,10 @@
21      B.addAttribute(llvm::Attribute::Cold);
22  }
23
24  + //HACK: maybe there is a better way to do this
25  + if (const ObfKeyAttr *OKA = D->getAttr<ObfKeyAttr>())
26  +   B.addAttribute("ObfuscationKey", OKA->getKey());
27  +
28  + if (D->hasAttr<MinSizeAttr>())
29  +   B.addAttribute(llvm::Attribute::MinSize);
30
31  --- tools/clang/lib/Sema/SemaDeclAttr.cpp        (revision 192535)
32  +++ tools/clang/lib/Sema/SemaDeclAttr.cpp        (working copy)
33  @@ -2759,6 +2759,17 @@
34
35      ParmType,
36      ↪Attr.getLoc()));
37  }
38
39  +static void handleObfKeyAttr(Sema &S, Decl *D, const AttributeList
40  +    ↪&Attr) {
41  +  // Make sure that there is a string literal as the sections's
42  +  ↪single
43  +  // argument.
44  +  StringRef Str;
45  +  SourceLocation LiteralLoc;
46  +  if (!S.checkStringLiteralArgumentAttr(Attr, 0, Str, &LiteralLoc))
47  +    return;
48  +
49  +  D->addAttr(::new (S.Context) ObfKeyAttr(Attr.getLoc(), S.Context,

```

```

    ↪Str, Attr.getAttributeSpellingListIndex()));
44 +}
45 +
46 SectionAttr *Sema::mergeSectionAttr(Decl *D, SourceRange Range,
47                                     StringRef Name,
48                                     unsigned AttrSpellingListIndex)
    ↪{
49 @@ -4646,6 +4657,7 @@
50     case AttributeList::AT_InitPriority:
51         handleInitPriorityAttr(S, D, Attr); break;
52
53 + case AttributeList::AT_ObfKey:         handleObfKeyAttr      (S, D,
    ↪Attr); break;
54     case AttributeList::AT_Packed:         handlePackedAttr      (S, D,
    ↪Attr); break;
55     case AttributeList::AT_Section:         handleSectionAttr      (S, D,
    ↪Attr); break;
56     case AttributeList::AT_Unavailable:

```

C.3 Obf library

src/Obf/Utils.h

```

1
2 #ifndef LLVM_OBF_UTILS_H
3 #define LLVM_OBF_UTILS_H
4 #include "llvm/IR/Function.h"
5 #include "llvm/IR/Module.h"
6 #include "llvm/ADT/StringRef.h"
7 #include "llvm/Support/CommandLine.h"
8 #include <algorithm>
9 #include <cstdlib>
10 #include <cstring>
11 #include <stdint>
12 #include "aes.h"
13
14
15 namespace Obf {
16     //Utilities for the Obfuscation transformations
17     //These involves mainly things like randomness generators and
18     ↪vector randomization
19
20     //This is the base class of a PRNG, includes some interesting
21     ↪functions
22     class PRNG_base {
23     protected:
24         //Minimal base implementation, generates a string of data
25         virtual void get_randoms(char *data, size_t len) = 0;
26     public:
27         virtual ~PRNG_base() {}
28         //Get a random integer
29         template <class int_t> int_t get_randomi(int_t end) {
30             int_t res;
31             get_randoms((char *)&res, sizeof(int_t));
32             res %= end;
33             //Negative modulus need to be normalized
34             res = abs(res);
35             return res;
36         }
37         template <class int_t> bool get_randomb(int_t num, int_t den)
38     ↪ {
39             int_t rnd = get_randomi(den);
40             bool rv = rnd < num;
41             return rv;
42         }
43         template <class int_t> int_t get_randomr(int_t begin, int_t
44     ↪end) {
45             return begin + (get_randomi(end-begin));
46         }
47     };
48 }

```



```

42     }
43     uint64_t rand64() {
44         return get_randomi((uint64_t)UINT64_MAX);
45     }
46     //Randomly rearrange the elements of a vector, uses swaps
47     ↪when available
48     template <class RandomAccessIterator> void randomize_vector(
49     ↪RandomAccessIterator first, RandomAccessIterator last) {
50         RandomAccessIterator rfirst = first;
51         while (last!=first) {
52             RandomAccessIterator relem = get_randomr(first,last);
53             assert(rfirst <= first && first < last && rfirst <=
54             ↪relem && first <= relem && relem <= last);
55             if (first != relem)
56                 std::swap(*first, *relem);
57             first++;
58         }
59     };
60     //Don't use, it is weak!
61     class PRNG_rand : public PRNG_base {
62     protected:
63         virtual void get_randoms(char *data, size_t len);
64     public:
65         virtual ~PRNG_rand() {}
66     };
67     class CPRNG_AES_CTR : public PRNG_base {
68     aes_encrypt_ctx cx;
69     unsigned char iv[AES_BLOCK_SIZE];
70     protected:
71         virtual void get_randoms(char *data, size_t len);
72     public:
73         CPRNG_AES_CTR (const llvm::Function &F, llvm::StringRef gref)
74     ↪;
75         CPRNG_AES_CTR (const llvm::Module &M, llvm::StringRef gref);
76         static llvm::StringRef get_obf_key(const llvm::Function &F);
77         static llvm::StringRef get_obf_key(const llvm::Module &M);
78         static void set_obf_key(llvm::Function &F, llvm::StringRef
79     ↪key);
80         static void set_obf_key(llvm::Module &M, llvm::StringRef key)
81     ↪;
82         static bool has_obf_key(const llvm::Function &F) {
83             return !get_obf_key(F).empty();
84         }
85         static bool has_obf_key(const llvm::Module &M) {
86             return !get_obf_key(M).empty();
87         }
88     };

```

```

85         virtual ~CPRNG_AES_CTR() {}
86     };
87
88     class Probability {
89     public:
90         uint64_t num;
91         uint64_t den;
92     public:
93         Probability() : num(0), den(1) {}
94         Probability(uint64_t num, uint64_t den) : num(num), den(den)
↪{
95         }
96         inline void set(uint64_t num, uint64_t den) {
97             this->num = num; this->den = den;
98         }
99         inline bool roll(PRNG_base &prng) const {
100             return prng.get_randomb(num, den);
101         }
102         inline bool rolldiv(PRNG_base &prng, uint64_t div) const {
103             return prng.get_randomb(num, den*div);
104         }
105     };
106     struct ProbabilityParser : public llvm::cl::parser<Probability> {
107         // parse - Return true on error.
108         bool parse(llvm::cl::Option &O, llvm::StringRef ArgName, const
↪std::string &ArgValue, Probability &Val);
109     };
110
111 };
112 #endif

```

src/Obf/Utils.cpp

```

1  #include <cinttypes>
2  #include <cstring>
3  #include "Utils.h"
4  #include "cmac.h"
5  #include "llvm/IR/Metadata.h"
6  #include "llvm/IR/Attributes.h"
7  #include "llvm/Support/raw_ostream.h"
8
9  //Utilities for the Obfuscation transformations
10 //These involve mainly things like randomness generators and vector
↪randomization
11 namespace Obf {
12     #define emptyStringRef llvm::StringRef()
13     static const char * ObfKeyMDName = "ObfuscationKey";
14     static const char * ObfKeyAttrName = "ObfuscationKey";
15     static const unsigned char nchar = '\0';
16     //Create a key for use with the tag generation algorithm

```

```

17      //This is CMAC_cmackey(kid//keydata) where kid is the key type (1
    ↪ for function 2 for modules) and keydata the keydata
18
19      static void make_cmac_key(unsigned char kid, llvm::StringRef
    ↪keydata, unsigned char*key) {
20          assert(!keydata.empty() && "The obfuscation key shouldn't be
    ↪empty");
21          const static unsigned char cmac_key[16] = {0xab,0xad,0xce,0
    ↪xba,0xda,0xbe,0xbe,0xfa,0xba,0xda,0xac,0xab,0xac,0xab,0xec,0xea};
22          cmac_ctx ctx;
23          cmac_init (cmac_key,&ctx);
24          cmac_data (&kid,1,&ctx);
25          cmac_data ((const unsigned char *)keydata.data(),keydata.size
    ↪(),&ctx);
26          cmac_end (key,&ctx);
27      }
28
29      static void make_zero_iv (unsigned char*iv) {
30          memset(iv,0,AES_BLOCK_SIZE);
31      }
32
33      static void make_tag(unsigned char tid, const unsigned char *key,
    ↪ llvm::StringRef mname, llvm::StringRef fname, llvm::StringRef gref,
    ↪ unsigned char *tag) {
34          cmac_ctx ctx;
35          //Get the key
36          cmac_init ((const unsigned char *)key,&ctx);
37          cmac_data (&tid,1,&ctx);
38          cmac_data ((const unsigned char *)mname.data(),mname.size(),&
    ↪ctx);
39          cmac_data (&nchar,1,&ctx);
40          if (!fname.empty()) {
41              cmac_data ((const unsigned char *)fname.data(),fname.size
    ↪(),&ctx);
42              cmac_data (&nchar,1,&ctx);
43          }
44          assert(!gref.empty() && "The transformation tag shouldn't be
    ↪empty");
45          cmac_data ((const unsigned char *)gref.data(),gref.size(),&
    ↪ctx);
46          cmac_data (&nchar,1,&ctx);
47          cmac_end (tag,&ctx);
48      }
49
50      llvm::StringRef CPRNG_AES_CTR::get_obf_key(const llvm::Function &
    ↪F) {
51          if(!F.hasFnAttribute(ObfKeyAttrName))
52              return emptystringref;
53          llvm::Attribute attr = F.getFnAttribute(ObfKeyAttrName);

```

```

54         if (!attr.isStringAttribute())
55             return emptystringref;
56         return attr.getValueAsString();
57     }
58
59     void CPRNG_AES_CTR::set_obf_key(llvm::Function &F, llvm::
↪StringRef key){
60         //Replace it
61         F.addFnAttr(ObfKeyAttrName, key);
62     }
63
64     llvm::StringRef CPRNG_AES_CTR::get_obf_key(const llvm::Module &M)
↪{
65         llvm::NamedMDNode *nm = M.getNamedMetadata(ObfKeyMDName);
66         if (!nm || nm->getNumOperands() != 1)
67             return emptystringref;
68         llvm::MDNode *md = nm->getOperand(0);
69         if (!md || md->getNumOperands() != 1)
70             return emptystringref;
71         llvm::MDString *mds = llvm::dyn_cast_or_null<llvm::MDString
↪>(md->getOperand(0));
72         if (!mds)
73             return emptystringref;
74         return mds->getString();
75     }
76
77     void CPRNG_AES_CTR::set_obf_key(llvm::Module &M, llvm::StringRef
↪key){
78         //First we have to delete the current metadata
79         llvm::NamedMDNode *om = M.getNamedMetadata(ObfKeyMDName);
80         if (om) {
81             M.eraseNamedMetadata(om);
82         }
83         llvm::NamedMDNode *nm = M.getOrInsertNamedMetadata(
↪ObfKeyMDName);
84         assert(nm && "Named_Metadadata_node_not_created");
85         assert(nm->getNumOperands() == 0 && "Named_Metadadata_node_not_
↪deleted");
86         llvm::MDString *mds = llvm::MDString::get(M.getContext(), key
↪);
87         assert(mds && "MDString_not_created");
88         llvm::MDNode *md = llvm::MDNode::get(M.getContext(), llvm::
↪ArrayRef<llvm::Value *>(mds));
89         assert(md && "MDNode_not_created");
90         nm->addOperand(md);
91     }
92
93     //Create a AES_CTR CPRNG object for use in a function
94     //The encryption key is created hashing with the CMAC algorithm:

```

```

95     //1||ModuleName||0||FunctionName||0||ObfModuleName||0
96     CPRNG_AES_CTR::CPRNG_AES_CTR (const llvm::Function &F, llvm::
    ↪StringRef gref) {
97         unsigned char nk[16];
98         unsigned char ck[16];
99         llvm::StringRef ok = get_obf_key(F);
100         assert(!ok.empty() && "No_obfuscation_key_found");
101         llvm::StringRef mname = F.getParent()->getModuleIdentifier();
102         llvm::StringRef fname = F.getName();
103         make_cmac_key(1,ok,ck);
104         make_tag(1,ck,mname,fname,gref,nk);
105         aes_encrypt_key128(nk,&cx);
106         make_zero_iv(iv);
107     }
108
109     //Create a AES_CTR CPRNG object for use in a module
110     //The encryption key is created hashing with the CMAC algorithm:
111     //2||ModuleName||0||ObfModuleName||0
112     CPRNG_AES_CTR::CPRNG_AES_CTR (const llvm::Module &M, llvm::
    ↪StringRef gref) {
113         unsigned char nk[16];
114         unsigned char ck[16];
115         llvm::StringRef ok = get_obf_key(M);
116         assert(!ok.empty() && "No_obfuscation_key_found");
117         llvm::StringRef mname = M.getModuleIdentifier();
118         llvm::StringRef fname = emptystringref;
119         make_cmac_key(2,ok,ck);
120         make_tag(2,ck,mname,fname,gref,nk);
121         aes_encrypt_key128(nk,&cx);
122         make_zero_iv(iv);
123     }
124
125     void PRNG_rand::get_randoms(char *data, size_t len) {
126         for(size_t i=0; i < len ; i++) {
127             data[i]=rand();
128         }
129     }
130
131     //We can generate up to 16 bytes of random data per call, we
    ↪generate only half of them to make
132     //finding the key or the plain text harder in the unlikely case
    ↪AES is broken
133     void CPRNG_AES_CTR::get_randoms(char *data, size_t len) {
134         size_t i = 0;
135         while( i < len ) {
136             unsigned char buf[AES_BLOCK_SIZE];
137             AES_RETURN rv;
138             rv = aes_ctr_pad(buf, iv, &cx);
139             assert(rv == EXIT_SUCCESS && "Failure_generating_pseudo_

```

```

    ↪random_data");
140         if (len-i < 8)
141             memcpy(data+i,buf,len-i);
142         else
143             memcpy(data+i,buf,8);
144         i+=8;
145     }
146 }
147
148 bool ProbabilityParser::parse(llvm::cl::Option &O, llvm::
↪StringRef ArgName, const std::string &ArgValue, Probability &Val) {
149     int nchars;
150     uint64_t num, den;
151     int rv = sscanf(ArgValue.c_str(), "%" PRIu64 "/" PRIu64 "%n"
↪,&num,&den,&nchars);
152     if (rv != 2 || nchars != (int)ArgValue.size())
153         return O.error("'" + ArgValue + "'_used_in_" + ArgName +
↪ "'_is_not_a_valid_probability!");
154     Val.set(num,den);
155     return false;
156 }
157 };

```

src/Obf/AddModuleKey.cpp

```

1  #define DEBUG_TYPE "addmodulekey"
2  #include "llvm/IR/Module.h"
3  #include "llvm/Support/CommandLine.h"
4  #include "llvm/Pass.h"
5  #include "llvm/Support/ErrorHandler.h"
6  #include "llvm/Support/raw_ostream.h"
7  #include "Utils.h"
8  using namespace llvm;
9  using namespace Obf;
10
11 namespace {
12     //TODO: generate a random key when none is specified
13     static cl::opt< std::string > TheKey ("modulekey", cl::desc("
↪Specify the module obfuscation key"), cl::value_desc("obfkey"), cl::
↪Optional);
14     struct AddModuleKey : public ModulePass {
15         static char ID; // Pass identification, replacement for
↪typeid
16         AddModuleKey() : ModulePass(ID) {}
17         virtual bool runOnModule(Module &M){
18             if (TheKey.getNumOccurrences() != 1)
19                 TheKey.error("This option has to be declared when
↪using the addmodulekey pass");
20             if (TheKey.empty())
21                 TheKey.error("No key (or an empty key) was defined,

```

```

    ↪set_some_key");
22         CPRNG_AES_CTR::set_obf_key(M, TheKey);
23         return true;
24     }
25 };
26 }
27
28 char AddModuleKey::ID = 0;
29 static RegisterPass<AddModuleKey> X("addmodulekey", "Add the desired
    ↪obfuscation key to the module requires the -modulekey<modulekey>
    ↪option set");

```

src/Obf/BBSplit.cpp

```

1  #define DEBUG_TYPE "bbsplit"
2  #include "llvm/ADT/Statistic.h"
3  #include "llvm/IR/InstrTypes.h"
4  #include "llvm/IR/Instruction.h"
5  #include "llvm/IR/BasicBlock.h"
6  #include "llvm/IR/Function.h"
7  #include "llvm/IR/User.h"
8  #include "llvm/Pass.h"
9  #include "llvm/Transforms/Utils/BasicBlockUtils.h"
10 #include "llvm/Analysis/LoopInfo.h"
11 #include "llvm/Analysis/Dominators.h"
12 #include "llvm/ADT/Twine.h"
13 #include "Utils.h"
14 #include <vector>
15 using namespace llvm;
16
17 STATISTIC(BBSplitCounter, "Number of basic blocks splitted");
18
19
20 namespace {
21     static Obf::Probability initialProbability(1,16);
22     static cl::opt< Obf::Probability, false, Obf::ProbabilityParser >
    ↪ splitProbability ("splitprobability", cl::desc("Specify the
    ↪probability of splitting a BB"), cl::value_desc("probability"), cl::
    ↪Optional, cl::init(initialProbability));
23     typedef std::vector<BasicBlock*> blist;
24     struct BBSplit : public FunctionPass {
25         static char ID; // Pass identification, replacement for
    ↪typeid
26         BBSplit() : FunctionPass(ID) {
27         }
28
29         virtual bool runOnFunction(Function &F) {
30             //if no module key found just leave the function alone
31             if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
32                 return false;

```

```

33
34         Obf::CPRNG_AES_CTR prng(F, "bbsplit");
35         bool rval = false;
36         blist BBlist;
37         BBlist.reserve(F.size());
38         //Fill the vector to prevent iterator invalidation
39         for (Function::iterator B = F.begin(); B != F.end(); B++)
40             BBlist.push_back(B);
41         for (blist::iterator B = BBlist.begin(); B != BBlist.end
↳(); B++) {
42             BasicBlock * cbb = *B;
43             unsigned splitcnt=1;
44             //We go to the instruction after the first (if any)
45             for (BasicBlock::iterator I=((*B)->
↳getFirstInsertionPt())++; I != cbb->end(); I++) {
46                 if (splitProbability.roll(prng)) {
47                     cbb=SplitBlock(cbb, I, this);
48                     cbb->setName(Twine((*B)->getName(), ".rsplit")
↳+Twine(splitcnt));
49                     //Again get then next instruction after the
↳one where we did the split
50                     I=(cbb->getFirstInsertionPt())++;
51                     rval=true;
52                     splitcnt++;
53                     ++BBSplitCounter;
54                 }
55             }
56         }
57         return rval;
58     }
59     void getAnalysisUsage(AnalysisUsage &AU) const {
60         AU.addPreserved<LoopInfo>();
61         AU.addPreserved<DominatorTree>();
62     }
63 };
64 }
65
66 char BBSplit::ID = 0;
67 static RegisterPass<BBSplit> X("bbsplit", "Randomly split basic
↳blocks into two");

```

src/Obf/FlattenControl.cpp

```

1 #define DEBUG_TYPE "flattencontrol"
2 #include "llvm/IR/InstrTypes.h"
3 #include "llvm/IR/Instruction.h"
4 #include "llvm/IR/Instructions.h"
5 #include "llvm/IR/BasicBlock.h"
6 #include "llvm/IR/Function.h"
7 #include "llvm/IR/User.h"

```



```

8  #include "llvm/Pass.h"
9  #include "llvm/IR/Constants.h"
10 #include "llvm/Transforms/Utils/UnifyFunctionExitNodes.h"
11 #include "llvm/ADT/Twine.h"
12 #include "llvm/ADT/SmallPtrSet.h"
13 #include "llvm/ADT/SmallVector.h"
14 #include "llvm/ADT/DenseMap.h"
15 #include "llvm/Transforms/Utils/BasicBlockUtils.h"
16 #include "Utils.h"
17 #include <vector>
18 #include <cstdint>
19 #include "llvm/Support/raw_ostream.h"
20 using namespace llvm;
21 using namespace Obf;
22
23 namespace {
24     typedef SmallVector<BasicBlock*,128> bblist;
25     typedef std::vector<Use*> uselist;
26     typedef SmallPtrSet<BasicBlock*,128> bbset;
27     typedef SmallDenseMap<BasicBlock*, ConstantInt*, 128> bb2id;
28     struct FlattenControl : public FunctionPass {
29         static char ID; // Pass identification, replacement for
↪typeid
30         FlattenControl() : FunctionPass(ID) {}
31         void generateBBIDs(Function &F, bblist &sbbs, bb2id &bbids,
↪PRNG_base &prng) const {
32             //This function generates an unique identifier for each
↪BB with incoming branches
33             //The identifiers follow a set of properties to make the
↪mainblock jump more efficient
34             //In particular they go from 0 to the number of blocks-1
↪to which jumps are possible
35             // so tables can be compact
36             //Works better if called before creating the mainblock
↪itself
37             //Step 1 create a set with all the target bbs we can
↪reach
38             bbset bbs;
39             bblist bbsv;
40             for (bblist::iterator B = sbbs.begin(), e = sbbs.end(); B
↪ != e; B++) {
41                 TerminatorInst *t = (*B)->getTerminator();
42                 for (unsigned i = 0, e = t->getNumSuccessors(); i !=
↪e; i++) {
43                     bbs.insert(t->getSuccessor(i));
44                 }
45             }
46             //Convert it into a vector
47             bbsv.reserve(bbs.size());

```

```

48         for (bbset::iterator i=bbs.begin(),e=bbs.end(); i != e; i
↳++) {
49             bbsv.push_back(*i);
50         }
51         //Randomize it
52         prng.randomize_vector(bbsv.begin(),bbsv.end());
53         //And finally associate the element position to each
↳block on the bb2id
54         int32_t id = 0;
55         for (bblist::iterator i=bbsv.begin(),e=bbsv.end(); i != e
↳; i++) {
56             bbids[*i]=ConstantInt::get(F.getContext(), APInt(32,
↳id));
57             id++;
58         }
59         return;
60     }
61     virtual bool runOnFunction(Function &F) {
62         //if no module key found just leave the function alone
63         if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
64             return false;
65
66         CPRNG_AES_CTR prng(F,"flattencontrol");
67         bblist BranchBlocks;
68         //Ensure our entry point contains only the branch
↳instruction
69         BasicBlock* newentry = &(F.getEntryBlock());
70         {
71             BasicBlock* entry = newentry;
72             TerminatorInst *t = entry->getTerminator();
73             BranchInst *BI = dyn_cast<BranchInst>(t);
74             if (&(entry->getFirstInsertionPt()) != t || !BI ||
↳BI->isConditional()) {
75                 newentry = BasicBlock::Create(F.getContext(), "
↳newentry",&F,entry);
76                 BranchInst::Create(entry,newentry);
77             }
78         }
79         //The blocks we will process, this ensures iterators don'
↳t break entry is included
80         BranchBlocks.reserve(F.size()); //Number of blocks + the
↳new entry block
81         for (Function::iterator B = F.begin(); B != F.end(); B++)
82             BranchBlocks.push_back(B);
83         UnifyFunctionExitNodes &UFEN = getAnalysis<
↳UnifyFunctionExitNodes>();
84         //Create the unreachable block for the switch (if one isn
↳'t already there)
85         BasicBlock* unr = UFEN.getUnreachableBlock();

```

```

86         if (!unr) {
87             //If we create this node we don't want it on the list
            ↪ processed by the algorithm hence the position
88             unr = BasicBlock::Create(F.getContext(), "
            ↪UnifiedUnreachableBlock", &F);
89             new UnreachableInst(F.getContext(), unr);
90         }
91         BasicBlock* main_node = BasicBlock::Create(F.getContext()
            ↪, "mainblock",&F,++Function::iterator(newentry));
92         //Used later, moved here for efficiency
93         { //Keep the live of the list limited
94             uselist ul;
95             //Check all the uses of the operands, if they are
            ↪instructions outside of our basic block or the main block or are
            ↪phis
96             //we add a phi for them on the main block so uses
            ↪dominate users :)
97             for(bblast::iterator Bi = BranchBlocks.begin(); Bi !=
            ↪ BranchBlocks.end(); Bi++) {
98                 BasicBlock *B = *Bi;
99                 TerminatorInst *TI=B->getTerminator();
100                 for(BasicBlock::iterator It = B->begin(); It != B
            ↪->end(); It++) {
101                     //Generate a list of the uses we are
            ↪interested in
102                     //These are non phi uses outside of the BB
            ↪and the BB terminator
103                     bool keepvalues = false;
104                     ul.reserve(It->getNumUses()); //Ensure enough
            ↪space is available
105                     for (Value::use_iterator Ut = It->use_begin()
            ↪; Ut != It->use_end(); Ut++) {
106                         Use *U = &(Ut.getUse());
107                         Instruction *I = dyn_cast<Instruction>(U
            ↪->getUser());
108                         if (I == 0) // Not a instruction so we
            ↪don't care
109                             continue;
110                         //It is a PHINode, these are handled in a
            ↪different way
111                         if (isa<PHINode>(*I)) {
112                             PHINode * pn = cast<PHINode>(I);
113                             //We only want to ignore it if it
            ↪refers to this block (so the instruction will be used instead)
114                             if (pn->getIncomingBlock(*U) == B)
115                                 continue;
116                             keepvalues = true;
117                         } else
118                             //If we refer to it from another block we

```

```

119         if (I->getParent() != B)
120             keepvalues = true;
121         else if (I != TI || isa<BranchInst>(TI))
122             // Same block and not the terminator, ignore the instruction
123             continue;
124         ul.push_back(U);
125     }
126     //There is at least one interesting use
127     if (!ul.empty()) {
128         Type *type = It->getType();
129         UndefValue *undef = UndefValue::get(type)
130     };
131     //TODO optimize so it won't always keep
132     //the variables
133     PHINode *phi = PHINode::Create(type,
134     BranchBlocks.size(), Twine(It->getName(), ".uses"), main_node);
135     Value *def = undef; //The default value,
136     //if no need to keep it it will be undefined
137     if (keepvalues)
138         def = phi; //Keep the value
139     for (bblast::iterator Bj = BranchBlocks.
140     begin(); Bj != BranchBlocks.end(); Bj++) {
141         BasicBlock *Bjp = *Bj;
142         if (B == Bjp)
143             phi->addIncoming(&*It, Bjp); //
144     //Assign the value
145         else if (Bjp == newentry)
146             phi->addIncoming(undef, Bjp); //
147     //Undefined if it comes from the entry
148         else
149             phi->addIncoming(def, Bjp); //
150     //Keep or undef
151     }
152     //Replace the uses
153     for (uselist::iterator U = ul.begin(); U
154     != ul.end(); U++) {
155         (*U)->set(phi);
156     }
157     ul.clear();
158 }
159 }
160 }
161 }
162 //Go through our block list moving phis to the core block
163 .
164 //Order of the phis doesn't matter as they always refer
165 //to the predecessor block variables

```

```

154         for(bblast::iterator Bi = BranchBlocks.begin(); Bi !=
↪BranchBlocks.end(); Bi++) {
155             BasicBlock *B = *Bi;
156             PHINode *newphi;
157             BasicBlock::iterator Ii=B->begin();
158             while (isa<PHINode>(*Ii)) {
159                 bbset oldbbs;
160                 PHINode *oldphi = cast<PHINode>(Ii);
161                 Type *type = oldphi->getType();
162                 newphi = PHINode::Create(type, BranchBlocks.size
↪(), "", main_node);
163                 //Take the name of the old phi
164                 newphi->takeName(oldphi);
165                 //Parse and move entries from the phi node (first
↪pass)
166                 for (User::op_iterator O = oldphi->op_begin(); O
↪!= oldphi->op_end(); O++) {
167                     BasicBlock *oldbb;
168                     oldbb = oldphi->getIncomingBlock(*O);
169                     newphi->addIncoming(*O,oldbb);
170                     oldbbs.insert(oldbb);
171                 }
172                 //Fill it the rest with keeps (second pass)
173                 for (bblast::iterator Bj = BranchBlocks.begin();
↪Bj != BranchBlocks.end(); Bj++) {
174                     BasicBlock *Bjp = *Bj;
175                     if (oldbbs.count(Bjp))
176                         continue;
177                     //TODO: we should improve this to make undef
↪if usage is impossible
178                     if (Bjp == newentry)
179                         newphi->addIncoming(UndefValue::get(type)
↪,Bjp); //Undefined if it comes from the entry
180                     else
181                         newphi->addIncoming(newphi, Bjp); //Keep
↪the value for future references
182                     }
183                     ReplaceInstWithValue(B->getInstList(), Ii, newphi
↪);
184                 }
185             }
186             //Split blocks with terminators we don't know how to
↪handle to get a br we know how to handle
187             for(bblast::iterator Bi = BranchBlocks.begin(); Bi !=
↪BranchBlocks.end(); Bi++) {
188                 BasicBlock *B = *Bi;
189                 TerminatorInst *t = B->getTerminator();
190                 //Split the non conditional branches
191                 if (!isa<BranchInst>(*t)) {

```

```

192         //TODO: it is better if we get as much of the
        ↪flow control as possible here
193         //Use faster function since the transformation
        ↪breaks the analysis anyways
194         B->splitBasicBlock(t, Twine(B->getName(), ".tflat")
        ↪);
195     }
196 }
197 //Generate the list of possible destinations for our
        ↪blocks
198     bb2id bbids;
199     generateBBIDs(F, BranchBlocks, bbids, prng);
200     PHINode *phi = PHINode::Create(IntegerType::get(F.
        ↪getContext(), 32), BranchBlocks.size(), "mainphi", main_node);
201     for(bblist::iterator Bi = BranchBlocks.begin(); Bi !=
        ↪BranchBlocks.end(); Bi++) {
202         Value *phiv;
203         BasicBlock *B = *Bi;
204         TerminatorInst *t = B->getTerminator();
205         BranchInst *BI = dyn_cast<BranchInst>(t);
206         if (BI && BI->isConditional()) {
207             //We associate a number with the destination
208             BasicBlock * s0 = BI->getSuccessor(0), *s1 = BI->
        ↪getSuccessor(1);
209             assert(bbids.count(s0) && "Successor_0_not_on_the
        ↪list");
210             assert(bbids.count(s1) && "Successor_1_not_on_the
        ↪list");
211             phiv = SelectInst::Create (BI->getCondition(),
        ↪bbids[s0], bbids[s1], Twine(B->getName(), ".br_select"), t);
212         } else {
213             BasicBlock * s = BI->getSuccessor(0);
214             assert(bbids.count(s) && "Successor_not_on_the_
        ↪list");
215             //We add the number to the PHI in the main_node
216             phiv=bbids[s];
217         }
218         //Add the value to the phi
219         phi->addIncoming(phiv, B);
220         //We make the block branch to the core block
221         ReplaceInstWithInst(t, BranchInst::Create(main_node))
        ↪;
222     }
223     //Now the switch instruction
224     SwitchInst *sw = SwitchInst::Create(phi, unr, bbids.size
        ↪(), main_node);
225     for (bb2id::iterator i=bbids.begin(), e=bbids.end(); i != e
        ↪; i++) {
226         sw->addCase(i->second, i->first);

```

```

227         }
228         return true;
229     }
230     void getAnalysisUsage(AnalysisUsage &AU) const {
231         AU.addRequired<UnifyFunctionExitNodes>(); //Passing this
→improves the resulting code a lot
232     }
233 };
234 }
235
236 char FlattenControl::ID = 0;
237 static RegisterPass<FlattenControl> X("flattencontrol", "Flatten_all_
→the_nodes_to_a_single_node_to_obfuscate_the_code");

```

src/Obf/ObfuscateConstants.cpp

```

1  #define DEBUG_TYPE "obfuscateconstants"
2  #include "llvm/IR/InstrTypes.h"
3  #include "llvm/IR/Instruction.h"
4  #include "llvm/IR/Instructions.h"
5  #include "llvm/IR/BasicBlock.h"
6  #include "llvm/IR/Function.h"
7  #include "llvm/IR/Module.h"
8  #include "llvm/IR/User.h"
9  #include "llvm/Pass.h"
10 #include "llvm/ADT/APInt.h"
11 #include "llvm/ADT/Statistic.h"
12 #include "llvm/IR/Constants.h"
13 #include <vector>
14 #include <cstdint>
15 #include <Utils.h>
16 #include "llvm/Support/raw_ostream.h"
17 using namespace llvm;
18
19 STATISTIC(ObfuscatedPHIs, "Number_of_phis_with_constants_obfuscated")
→;
20 STATISTIC(ObfuscatedIns, "Number_of_instructions_with_constants_
→obfuscated");
21 STATISTIC(ObfuscatedUses, "Number_of_constants_uses_obfuscated");
22 STATISTIC(ObfuscatedCons, "Number_of_constants_obfuscated");
23 STATISTIC(ReobfuscatedCons, "Number_of_constants_reobfuscated_
→obfuscated_after_obfuscation");
24
25 namespace {
26     static Obf::Probability initialProbability(1,10);
27     static cl::opt< Obf::Probability, false, Obf::ProbabilityParser >
→ reobfuscationProbability ("reobfuscationprobability", cl::desc("
→Specify_the_probability_of_obfuscating_again_a_constant"), cl::
→value_desc("probability"), cl::Optional, cl::init(initialProbability
→));

```

```

28     //The real pass putting it here makes some code simpler
29     class DoObfuscateConstants {
30         Module &M;
31         GlobalVariable *intC;
32         IntegerType *intTy;
33         PointerType *intTyPtr;
34         std::vector<Constant *> intVs;
35         unsigned typelength;
36         Obf::CPRNG_AES_CTR *prng;
37         inline bool runOnFunction(Function &F) {
38             //if no module key found just leave the function alone
39             if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
40                 return false;
41
42             bool rval = false;
43             prng = new Obf::CPRNG_AES_CTR(F, "obfuscateconstants");
44             for (Function::iterator B = F.begin(); B != F.end(); B++)
45                 {
46                     for (BasicBlock::iterator I=B->begin(); isa<PHINode
47                     ↪>(*I); I++)
48                         if(runOnPHI(*cast<PHINode>(&*I))) {
49                             ObfuscatedPHIs++;
50                             rval = true;
51                         }
52                     for (BasicBlock::iterator I=B->getFirstInsertionPt();
53                     ↪ I != B->end(); I++)
54                         if(runOnNonPHI(*I)) {
55                             ObfuscatedIns++;
56                             rval = true;
57                         }
58                 }
59             delete prng;
60             return rval;
61
62             /*obfuscate a constant by introducing instructions before the
63             ↪ insertionPoint*/
64             inline Value * obfuscateConstant (Constant &C, Instruction *
65             ↪insertBefore) {
66                 /*TODO:As of now we can only obfuscate these*/
67                 if(isa<ConstantInt>(C)) {
68                     ObfuscatedCons++;
69                     ConstantInt &IC = cast<ConstantInt>(C);
70                     if (IC.getType()->getBitWidth() <= typelength && prng
71                     ↪->get_randomb(1,2)) {
72                         //Obfuscation technique 1: search for the
73                         ↪ constant in a vector
74                         ConstantInt *Cptr = ConstantInt::get(intTy,intVs.
75                         ↪size());
76                         intVs.push_back(ConstantInt::get(intTy,IC.

```



```

    ↪getValue().zextOrSelf(typelength));
69         Value *Vptr = Cptr;
70         if (reobfuscationProbability.roll(*prng)) {
71             ReobfuscatedCons++;
72             Vptr = obfuscateConstant(*Cptr,insertBefore);
73         }
74         LoadInst *lic = new LoadInst(intC, "", false,
    ↪insertBefore);
75         GetElementPtrInst* ptr = GetElementPtrInst::
    ↪Create(lic, Vptr, "",insertBefore);
76         LoadInst *li = new LoadInst(ptr, "", false,
    ↪insertBefore);
77         if (IC.getType()->getBitWidth() == typelength)
78             return li;
79         else return new TruncInst(li, IC.getType(), "",
    ↪insertBefore);
80     } else {
81         //Obfuscation technique 2: replace constant by an
    ↪ addition or subtraction etc of two other constants
82         ConstantInt *C1 = ConstantInt::get(IC.getType(),
    ↪prng->rand64());
83         //Maybe keep obfuscating the new constant
84         Value *V1 = C1;
85         if (reobfuscationProbability.rolldiv(*prng,2)) {
86             ReobfuscatedCons++;
87             V1 = obfuscateConstant(*C1,insertBefore);
88         }
89         APInt VC2;
90         Instruction::BinaryOps op;
91         //Basic example, we only use Add sub or xor since
    ↪ muls ands and ors are more complicated
92         switch (prng->get_randomi(3)) {
93             case 0:
94                 VC2 = IC.getValue()-C1->getValue();
95                 op=Instruction::Add;
96                 assert((VC2 + C1->getValue())==IC.
    ↪getValue());
97                 break;
98             case 1:
99                 VC2 = IC.getValue()+C1->getValue();
100                 op=Instruction::Sub;
101                 assert((VC2 - C1->getValue())==IC.
    ↪getValue());
102                 break;
103             case 2:
104                 VC2 = IC.getValue()^C1->getValue();
105                 op=Instruction::Xor;
106                 assert((VC2 ^ C1->getValue())==IC.
    ↪getValue());

```

```

107         break;
108     }
109     ConstantInt *C2 = cast<ConstantInt>(ConstantInt::
→get(IC.getType(),VC2));
110     Value *V2 = C2;
111     //Maybe keep obfuscating the new constant
112     if (reobfuscationProbability.rolldiv(*prng,2)) {
113         ReobfuscatedCons++;
114         V2 = obfuscateConstant(*C2,insertBefore);
115     }
116     return BinaryOperator::Create(op, V2, V1, "",
→insertBefore);
117 }
118 //TODO: obfuscation technique 3: use a formula
119 →returning the constant over some previous value
120     }
121     return &C;
122 }
123 //Obfuscate an Use if it s a constant (and we want to do so)
124 //Returns true if the use was modified
125 inline bool obfuscateUse(Use &U, Instruction *insertBefore) {
126     Constant *C = dyn_cast<Constant>(U.get());
127     if (C == 0) return false; //Not a constant
128     Value *NC = obfuscateConstant(*C, insertBefore);
129     if (NC == C) return false; //The constant wasn't modified
130     ObfuscatedUses++;
131     U.set(NC);
132     return true;
133 }
134 /* Run on a phi instruction */
135 inline bool runOnPHI(PHINode &phi) {
136     bool rval = false;
137     /*If a constant is found the value must be calculated on
138 →the phy node bringing us here*/
139     for (User::op_iterator O = phi.op_begin(); O != phi.
→op_end(); O++) {
140         rval |= obfuscateUse(*O,phi.getIncomingBlock(*O)->
→getTerminator());
141     }
142     return rval;
143 }
144 /* Run on a non phi instruction*/
145 inline bool runOnNonPHI(Instruction &I) {
146     bool rval=false;
147     /*Check only value (arg 1)*/
148     if(isa<SwitchInst>(I))
149         return obfuscateUse(I.getOperandUse(0),&I);
150     /*Check only vectors (args 1 and 2)*/
151     if(isa<ShuffleVectorInst>(I))

```

```

150         return obfuscateUse(I.getOperandUse(0),&I) |
↪obfuscateUse(I.getOperandUse(1),&I);
151         /*Check only struct and value (args 1 and 2)*/
152         if(isa<InsertValueInst>(I))
153             return obfuscateUse(I.getOperandUse(0),&I) |
↪obfuscateUse(I.getOperandUse(1),&I);
154         /*Check only struct (arg 1)*/
155         if(isa<ExtractValueInst>(I))
156             return obfuscateUse(I.getOperandUse(0),&I);
157         /*Check only NumElements (arg 1)*/
158         if(isa<AllocaInst>(I))
159             return obfuscateUse(I.getOperandUse(0),&I);
160         /*Ignore alignment*/
161         if(isa<LoadInst>(I))
162             return obfuscateUse(I.getOperandUse(0),&I);
163         /*TODO: Ignore constants in structs*/
164         if(isa<GetElementPtrInst>(I))
165             return false;
166         /*landingpads??*/
167         /*Intrinsics some lifetime ie give problems*/
168         if(isa<CallInst>(I))
169             return false;
170         /*Check all the values*/
171         for (User::op_iterator O = I.op_begin(); O != I.op_end();
↪ O++) {
172             rval |= obfuscateUse(*O,&I);
173         }
174         return rval;
175     }
176     public:
177     DoObfuscateConstants(Module &M) : M(M) {
178     }
179     bool run() {
180         //TODO: This should depend on the target type
181         typelength=64;
182         intTy = IntegerType::get(M.getContext(), typelength);
183         intTyPtr = PointerType::get(intTy, 0);
184
185         intC = new GlobalVariable(M,intTyPtr,false,GlobalVariable
↪::PrivateLinkage,0,".data");
186         bool rval = false;
187         for (Module::iterator F = M.begin(); F != M.end(); F++) {
188             if (F->empty())
189                 continue;
190             rval |= runOnFunction(*F);
191         }
192         //TODO: check deeply the possibility of getting rid of
↪the pointer memory access by using a placeholder
193         ArrayType* ArrayTy = ArrayType::get(intTy, intVs.size());

```

```

194         GlobalVariable *arrC = new GlobalVariable(M, ArrayTy, false
→, GlobalVariable::PrivateLinkage, ConstantArray::get(ArrayTy, intVs));
195         intC->setInitializer(ConstantExpr::getGetElementPtr(arrC,
→ std::vector<Constant*>(2, ConstantInt::get(intTy, 0))));
196         return rval;
197     }
198 };
199 //Saddly we can't keep global state if using the FunctionPass :(
200     struct ObfuscateConstants : public ModulePass {
201         static char ID; // Pass identification, replacement for
→ typeid
202         ObfuscateConstants() : ModulePass(ID) {}
203         virtual bool runOnModule(Module &M){
204             DoObfuscateConstants obc(M);
205             return obc.run();
206         }
207     };
208 }
209
210 char ObfuscateConstants::ID = 0;
211 static RegisterPass<ObfuscateConstants> X("obfuscateconstants", "
→Obfuscate the code constants by converting them into mathematical
→operations and dereferences from a vector");

```

src/Obf/PropagateModuleKey.cpp

```

1  #define DEBUG_TYPE "propagatemodulekey"
2  #include "llvm/IR/Module.h"
3  #include "llvm/IR/Function.h"
4  #include "llvm/Support/CommandLine.h"
5  #include "llvm/Pass.h"
6  #include "llvm/Support/raw_ostream.h"
7  #include "Utils.h"
8  using namespace llvm;
9  using namespace Obf;
10
11 namespace {
12     //TODO: add flag to decide whether overwrite keys, merge them or
→ keep them
13     //For now we just replace
14     struct PropagateModuleKey : public FunctionPass {
15         static char ID; // Pass identification, replacement for
→ typeid
16         PropagateModuleKey() : FunctionPass(ID) {}
17         virtual bool runOnFunction(Function &F) {
18             //if no module key found just leave it alone
19             if (!CPRNG_AES_CTR::has_obf_key(*(F.getParent())))
20                 return false;
21             StringRef TheKey = CPRNG_AES_CTR::get_obf_key(*(F.
→ getParent()));

```

```

22         CPRNG_AES_CTR::set_obf_key(F, TheKey);
23         return true;
24     }
25 };
26 }
27
28 char PropagateModuleKey::ID = 0;
29 static RegisterPass<PropagateModuleKey> X("propagatemodulekey", "
↳Propagate the module obfuscation key to the function");

src/Obf/RandBB.cpp
1  #define DEBUG_TYPE "randbb"
2  #include "llvm/IR/BasicBlock.h"
3  #include "llvm/IR/Function.h"
4  #include "llvm/IR/User.h"
5  #include "llvm/Pass.h"
6  #include "llvm/ADT/SmallVector.h"
7  #include <algorithm>
8  #include "Utils.h"
9  using namespace llvm;
10
11 namespace {
12     typedef SmallVector<BasicBlock*,128> candmap;
13     struct RandBB : public FunctionPass {
14         static char ID; // Pass identification, replacement for
↳typeid
15         RandBB() : FunctionPass(ID) {}
16
17         virtual bool runOnFunction(Function &F) {
18             //if no module key found just leave the function alone
19             if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
20                 return false;
21
22             Obf::CPRNG_AES_CTR prng(F,"randbb");
23             candmap candidates; //List of candidates
24             //Exclude the entry block
25             Function::iterator src=F.getEntryBlock();
26             candidates.reserve(F.size()-1);
27             //First pass, initialize structures
28             for (Function::iterator B = F.begin(), e = F.end(); B !=
↳e ; B++) {
29                 if (B != src) { //DO NOT MOVE THE ENTRY POINT!
30                     candidates.push_back(B);
31                 }
32             }
33             prng.randomize_vector(candidates.begin(),candidates.end()
↳);
34             for (candmap::size_type i = 0; i < candidates.size(); i
↳++) {

```

```

35         //Pick an element from the list
36         BasicBlock *dst=candidates[i];
37
38         if (dst != src) { //If swap is needed
39             dst->moveAfter(src);
40             src = dst;
41         }
42     }
43     return true;
44 }
45 void getAnalysisUsage(AnalysisUsage &AU) const {
46     AU.setPreservesCFG();
47 }
48 };
49 }
50
51 char RandBB::ID = 0;
52 static RegisterPass<RandBB> X("randbb", "Randomly rearrange BBs
↳inside functions keeping the entry point");

src/Obf/RandFun.cpp
1  #define DEBUG_TYPE "randfun"
2  #include "llvm/IR/Function.h"
3  #include "llvm/IR/Module.h"
4  #include "llvm/Pass.h"
5  #include "llvm/ADT/SmallVector.h"
6  #include <algorithm>
7  #include "llvm/Support/raw_ostream.h"
8  #include "Utils.h"
9  using namespace llvm;
10
11 namespace {
12     typedef SmallVector<Function*,128> candmap;
13     struct RandFun : public ModulePass {
14         static char ID; // Pass identification, replacement for
↳typeid
15         RandFun() : ModulePass(ID) {}
16
17         virtual bool runOnModule(Module &M) {
18             //if no module key found just leave the function alone
19             if (!Obf::CPRNG_AES_CTR::has_obf_key(M))
20                 return false;
21
22             Obf::CPRNG_AES_CTR prng(M,"randfun");
23             candmap candidates; //List of candidates
24             Module::iterator src=M.begin();
25             Module::FunctionListType &fl=M.getFunctionList();
26             candidates.reserve(M.size());
27             //First pass, initialize structures

```

```

28         for (Module::iterator F = src, e = M.end(); F != e ; F++)
↪ {
29             candidates.push_back(F);
30         }
31         prng.randomize_vector(candidates.begin(),candidates.end()
↪);
32         for (candmap::size_type i = 0; i < candidates.size(); i
↪++) {
33             //Pick an element from the list
34             Function *dst=candidates[i];
35
36             if (dst != src) { //If swap is needed
37                 fl.remove(dst);
38                 fl.insert(src,dst);
39             } else {
40                 src++;
41             }
42         }
43         return true;
44     }
45     void getAnalysisUsage(AnalysisUsage &AU) const {
46         AU.setPreservesCFG();
47     }
48 };
49 }
50
51 char RandFun::ID = 0;
52 static RegisterPass<RandFun> X("randfun", "Randomly rearrange
↪functions inside a module");

```

src/Obf/RandGlb.cpp

```

1  #define DEBUG_TYPE "randglb"
2  #include "llvm/IR/GlobalVariable.h"
3  #include "llvm/IR/Module.h"
4  #include "llvm/Pass.h"
5  #include "llvm/ADT/SmallVector.h"
6  #include <algorithm>
7  #include "llvm/Support/raw_ostream.h"
8  #include "Utils.h"
9  using namespace llvm;
10
11 namespace {
12     typedef SmallVector<GlobalVariable*,128> candmap;
13     struct RandGlb : public ModulePass {
14         static char ID; // Pass identification, replacement for
↪ typeid
15         RandGlb() : ModulePass(ID) {}
16
17         virtual bool runOnModule(Module &M) {

```

```

18         //if no module key found just leave the function alone
19         if (!Obf::CPRNG_AES_CTR::has_obf_key(M))
20             return false;
21
22         Obf::CPRNG_AES_CTR prng(M, "randglb");
23         candmap candidates; //List of candidates
24         Module::global_iterator src=M.global_begin();
25         Module::GlobalListType &gl=M.getGlobalList();
26         candidates.reserve(gl.size());
27         //First pass, initialize structures
28         for (Module::global_iterator G = src, e = M.global_end();
↪ G != e ; G++) {
29             candidates.push_back(G);
30         }
31         prng.randomize_vector(candidates.begin(), candidates.end()
↪ );
32         for (candmap::size_type i = 0; i < candidates.size(); i
↪ ++ ) {
33             //Pick an element from the list
34             GlobalVariable *dst=candidates[i];
35
36             if (dst != src) { //If swap is needed
37                 gl.remove(dst);
38                 gl.insert(src, dst);
39             } else {
40                 src++;
41             }
42         }
43         return true;
44     }
45     void getAnalysisUsage(AnalysisUsage &AU) const {
46         AU.setPreservesCFG();
47     }
48 };
49 }
50
51 char RandGlb::ID = 0;
52 static RegisterPass<RandGlb> X("randglb", "Randomly rearrange global
↪ variables inside a module");

```

src/Obf/RandIns.cpp

```

1 #define DEBUG_TYPE "randins"
2 #include "llvm/IR/Instructions.h"
3 #include "llvm/IR/Instruction.h"
4 #include "llvm/IR/BasicBlock.h"
5 #include "llvm/IR/Function.h"
6 #include "llvm/IR/User.h"
7 #include "llvm/Pass.h"
8 #include "llvm/ADT/DenseMap.h"

```



```

9  #include "llvm/ADT/SmallVector.h"
10 #include "llvm/ADT/SmallPtrSet.h"
11 #include <algorithm>
12 #include "Utils.h"
13 using namespace llvm;
14
15 namespace {
16     typedef SmallPtrSet<Instruction*,16> deplst;
17     typedef SmallDenseMap<Instruction*,deplst,256> depmap;
18     typedef SmallVector<Instruction*,256> candmap;
19     struct RandIns : public FunctionPass {
20         static char ID; // Pass identification, replacement for
→typeid
21         RandIns() : FunctionPass(ID) {}
22
23         virtual bool runOnFunction(Function &F) {
24             //if no module key found just leave the function alone
25             if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
26                 return false;
27
28             Obf::CPRNG_AES_CTR prng(F,"randins");
29             candmap candidates; //List of candidates
30             depmap dependencies;
31             unsigned dsti;
32             BasicBlock::iterator src;
33             for (Function::iterator B = F.begin(), e = F.end(); B !=
→e ; B++) {
34                 //First pass, initialize structures
35                 //Map each element to its position on the list (and
→the other way around)
36                 candidates.reserve(B->size());
37                 src=B->begin();
38                 for (BasicBlock::iterator I=src, e = BasicBlock::
→iterator(B->getFirstNonPHI()); I != e; I++) {
39                     candidates.push_back(I);
40                 }
41                 prng.randomize_vector(candidates.begin(),candidates.
→end());
42                 //Reorder Phi Nodes randomly
43                 for (candmap::size_type i = 0; i < candidates.size();
→ i++) {
44                     Instruction *dst=candidates[i];
45                     if (dst != src) { //If swap is needed
46                         dst->moveBefore(src);
47                     } else {
48                         src++;
49                     }
50                 }
51                 candidates.clear();

```

```

52         candidates.reserve(B->size());
53         dependencies.grow(B->size());
54         //Generate the dependency map
55         src= B->getFirstInsertionPt();
56         for (BasicBlock::iterator I=src, e = BasicBlock::
↪iterator(B->getTerminator()); I != e; I++) {
57             //Check the dependency map
58             deplst *lst = NULL;
59             //If the instruction may have undesired side
↪effects make it block other stuff with side effects or memory
↪accesses
60             if (I->mayHaveSideEffects()) for (BasicBlock::
↪iterator J=I; ++J != e;) {
61                 if (J->mayReadFromMemory() || J->
↪mayHaveSideEffects())
62                     dependencies[J].insert(I);
63             }
64             if (I->mayReadFromMemory()) for (BasicBlock::
↪iterator J=I; ++J != e;) {
65                 if (J->mayHaveSideEffects())
66                     dependencies[J].insert(I);
67             }
68             for (User::op_iterator U = I->op_begin(), e= I->
↪op_end(); U != e; ++U) {
69                 Instruction *i = dyn_cast<Instruction>(*U);
70                 if (!i)
71                     continue;
72                 if (isa<PHINode>(*i))
73                     continue;
74                 if (i->getParent() != B)
75                     continue;
76                 //Get the deplst
77                 if (!lst)
78                     lst = &(dependencies[I]);
79                 //Insert use on the map
80                 lst->insert(i);
81             }
82             if (!lst) {
83                 candidates.push_back(I);
84             }
85         }
86         //TODO: this should be done by the prng class given
↪the dependency list
87         //Reorder Instructions randomly
88         while (!candidates.empty()) {
89             //Pick a random element from the list
90             dsti = prng.get_randomi(candidates.size());
91             Instruction *dst=candidates[dsti];
92             if (dst != src) { //If swap is needed

```

```

93         dst->moveBefore(src);
94     } else {
95         src++;
96     }
97     candidates[dsti]=candidates.back();
98     candidates.pop_back();
99     //Remove the use from the dependencies
100     for (Value::use_iterator It = dst->use_begin(), e
↪ = dst->use_end(); It != e; ++It) {
101         Instruction *i = dyn_cast<Instruction>(*It);
102         depmap::iterator p = dependencies.find(i);
103         if ( p != dependencies.end() ) {
104             p->second.erase(dst);
105             if (p->second.empty()) {
106                 dependencies.erase(p);
107                 candidates.push_back(i);
108             }
109         }
110     }
111     dependencies.shrink_and_clear();
112     return true;
113 }
114 void getAnalysisUsage(AnalysisUsage &AU) const {
115     AU.setPreservesCFG();
116 }
117 };
118 }
119 }
120 }
121
122 char RandIns::ID = 0;
123 static RegisterPass<RandIns> X("randins", "Randomly rearrange
↪ instructions inside BBs keeping dependences");

```

src/Obf/SwapOps.cpp

```

1  #define DEBUG_TYPE "swapops"
2  #include "llvm/ADT/Statistic.h"
3  #include "llvm/IR/InstrTypes.h"
4  #include "llvm/IR/Instruction.h"
5  #include "llvm/IR/BasicBlock.h"
6  #include "llvm/IR/Function.h"
7  #include "llvm/IR/User.h"
8  #include "llvm/Pass.h"
9  #include "Utils.h"
10 using namespace llvm;
11
12 STATISTIC(SwapCounter, "Number of operands swapped");
13
14 namespace {

```

```

15     struct SwapOps : public FunctionPass {
16         static char ID; // Pass identification, replacement for
↪ typeid
17         SwapOps() : FunctionPass(ID) {}
18
19         virtual bool runOnFunction(Function &F) {
20             //if no module key found just leave the function alone
21             if (!Obf::CPRNG_AES_CTR::has_obf_key(F))
22                 return false;
23
24             bool rval = false;
25             Obf::CPRNG_AES_CTR prng(F, "swapops");
26             for (Function::iterator B = F.begin(); B != F.end(); B++)
↪ {
27                 for (BasicBlock::iterator I=B->getFirstInsertionPt();
↪ I != B->end(); I++) {
28                     if (!I->isCommutative())
29                         continue;
30                     BinaryOperator *BO = dyn_cast<BinaryOperator>(I);
31                     if (!BO)
32                         continue;
33                     if (prng.get_randomb(1,2)) {
34                         BO->swapOperands();
35                         rval=true;
36                         ++SwapCounter;
37                     }
38                 }
39             }
40             return rval;
41         }
42         void getAnalysisUsage(AnalysisUsage &AU) const {
43             AU.setPreservesCFG();
44         }
45     };
46 }
47
48 char SwapOps::ID = 0;
49 static RegisterPass<SwapOps> X("swapops", "Randomly swap the
↪ operators of commutative binary instructions");

```

C.4 AES library

src/Obf/aes.h

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24
25 This file contains the definitions required to use AES in C. See
26 ↪aesopt.h
27 for optimisation details.
28 */
29
30 #ifndef _AES_H
31 #define _AES_H
32
33 #include <stdlib.h>
34
35 /* This include is used to find 8 & 32 bit unsigned integer types
36 ↪*/
37 #include "brg_types.h"
38
39 #if defined(__cplusplus)
40 extern "C"
41 {
42 #endif
43
44 #define AES_128
45 #define FIXED_TABLES

```

```

39
40 /* The following must also be set in assembler files if being used
41 ↪*/
42 #define AES_ENCRYPT /* if support for encryption is needed
43 ↪*/
44 #define AES_BLOCK_SIZE 16 /* the AES block size in bytes
45 ↪*/
46 #define N_COLS 4 /* the number of columns in the state
47 ↪*/
48 /* The key schedule length is 11, 13 or 15 16-byte blocks for 128,
49 ↪*/
50 /* 192 or 256-bit keys respectively. That is 176, 208 or 240 bytes
51 ↪*/
52 /* or 44, 52 or 60 32-bit words.
53 ↪*/
54 #define KS_LENGTH 44
55 #define AES_RETURN INT_RETURN
56 /* the character array 'inf' in the following structures is used
57 ↪*/
58 /* to hold AES context information. This AES code uses cx->inf.b[0]
59 ↪*/
60 /* to hold the number of rounds multiplied by 16. The other three
61 ↪*/
62 /* elements can be used by code that implements additional modes
63 ↪*/
64
65 typedef union
66 {
67     uint_32t l;
68     uint_8t b[4];
69 } aes_inf;
70
71 typedef struct
72 {
73     uint_32t ks[KS_LENGTH];
74     aes_inf inf;
75 } aes_encrypt_ctx;
76
77 /* This routine must be called before first use if non-static
78 ↪*/
79 /* tables are being used
80 ↪*/
81
82 AES_RETURN aes_init(void);
83
84

```

```

75  /* Key lengths in the range 16 <= key_len <= 32 are given in bytes,
    ↪*/
76  /* those in the range 128 <= key_len <= 256 are given in bits
    ↪*/
77
78  AES_RETURN aes_encrypt_key128(const unsigned char *key,
    ↪aes_encrypt_ctx cx[1]);
79  AES_RETURN aes_encrypt(const unsigned char *in, unsigned char *out,
    ↪const aes_encrypt_ctx cx[1]);
80
81  /* Multiple calls to the following subroutines for multiple block
    ↪*/
82  /* ECB, CBC, CFB, OFB and CTR mode encryption can be used to handle
    ↪*/
83  /* long messages incrementally provided that the context AND the iv
    ↪*/
84  /* are preserved between all such calls. For the ECB and CBC modes
    ↪*/
85  /* each individual call within a series of incremental calls must
    ↪*/
86  /* process only full blocks (i.e. len must be a multiple of 16) but
    ↪*/
87  /* the CFB, OFB and CTR mode calls can handle multiple incremental
    ↪*/
88  /* calls of any length. Each mode is reset when a new AES key is
    ↪*/
89  /* set but ECB and CBC operations can be reset without setting a
    ↪*/
90  /* new key by setting a new IV value. To reset CFB, OFB and CTR
    ↪*/
91  /* without setting the key, aes_mode_reset() must be called and the
    ↪*/
92  /* IV must be set. NOTE: All these calls update the IV on exit so
    ↪*/
93  /* this has to be reset if a new operation with the same IV as the
    ↪*/
94  /* previous one is required (or decryption follows encryption with
    ↪*/
95  /* the same IV array).
    ↪*/
96
97  AES_RETURN aes_ecb_encrypt(const unsigned char *ibuf, unsigned char *
    ↪obuf, int len, const aes_encrypt_ctx ctx[1]);
98  AES_RETURN aes_ctr_pad(unsigned char *obuf, unsigned char *cbuf,
    ↪aes_encrypt_ctx cx[1]);
99
100 #if defined(__cplusplus)
101 }
102 #endif

```

```

103
104 #endif

src/Obf/aes__modes.c

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24
25 These subroutines implement multiple block AES modes for ECB, CBC,
26 ↪CFB,
27 OFB and CTR encryption, The code provides support for the VIA
28 ↪Advanced
29 Cryptography Engine (ACE).
30
31 NOTE: In the following subroutines, the AES contexts (ctx) must be
32 16 byte aligned if VIA ACE is being used
33 */
34
35 #include <string.h>
36 #include <assert.h>
37 #include <stdio.h>
38
39 #include "aesopt.h"
40
41 #if defined(__cplusplus)
42 extern "C"
43 {
44 #endif

```



```

39 #if defined( _MSC_VER ) && ( _MSC_VER > 800 )
40 #pragma intrinsic(memcpy)
41 #endif
42
43 #define BFR_BLOCKS      8
44
45 /* These values are used to detect long word alignment in order to */
46 /* speed up some buffer operations. This facility may not work on */
47 /* some machines so this define can be commented out if necessary */
48
49 #define FAST_BUFFER_OPERATIONS
50
51 #define lp32(x)          ((uint_32t*)(x))
52
53 #if defined( USE_VIA_ACE_IF_PRESENT )
54
55 #include "aes_via_ace.h"
56
57 #pragma pack(16)
58
59 aligned_array(unsigned long,    enc_gen_table, 12, 16) =
    ↪NEH_ENC_GEN_DATA;
60 aligned_array(unsigned long,    enc_load_table, 12, 16) =
    ↪NEH_ENC_LOAD_DATA;
61 aligned_array(unsigned long,    enc_hybrid_table, 12, 16) =
    ↪NEH_ENC_HYBRID_DATA;
62 aligned_array(unsigned long,    dec_gen_table, 12, 16) =
    ↪NEH_DEC_GEN_DATA;
63 aligned_array(unsigned long,    dec_load_table, 12, 16) =
    ↪NEH_DEC_LOAD_DATA;
64 aligned_array(unsigned long,    dec_hybrid_table, 12, 16) =
    ↪NEH_DEC_HYBRID_DATA;
65
66 /* NOTE: These control word macros must only be used after */
67 /* a key has been set up because they depend on key size */
68 /* See the VIA ACE documentation for key type information */
69 /* and aes_via_ace.h for non-default NEH_KEY_TYPE values */
70
71 #ifndef NEH_KEY_TYPE
72 #define NEH_KEY_TYPE NEH_HYBRID
73 #endif
74
75 #if NEH_KEY_TYPE == NEH_LOAD
76 #define kd_adr(c)      ((uint_8t*)(c)->ks)
77 #elif NEH_KEY_TYPE == NEH_GENERATE
78 #define kd_adr(c)      ((uint_8t*)(c)->ks + (c)->inf.b[0])
79 #elif NEH_KEY_TYPE == NEH_HYBRID
80 #define kd_adr(c)      ((uint_8t*)(c)->ks + ((c)->inf.b[0] == 160 ? 160
    ↪: 0))

```

```

81  #else
82  #error no key type defined for VIA ACE
83  #endif
84
85  #else
86
87  #define aligned_array(type, name, no, stride) type name[no]
88  #define aligned_auto(type, name, no, stride)  type name[no]
89
90  #endif
91
92  #if defined( _MSC_VER ) && _MSC_VER > 1200
93
94  #define via_cwd(cwd, ty, dir, len) \
95      unsigned long* cwd = (dir##_##ty##_table + ((len - 128) >> 4))
96
97  #else
98
99  #define via_cwd(cwd, ty, dir, len) \
100      aligned_auto(unsigned long, cwd, 4, 16); \
101      cwd[1] = cwd[2] = cwd[3] = 0; \
102      cwd[0] = neh_##dir##_##ty##_key(len)
103
104  #endif
105
106  AES_RETURN aes_ecb_encrypt(const unsigned char *ibuf, unsigned char *
↪obuf,
107                          int len, const aes_encrypt_ctx ctx[1])
108  {   int nb = len >> 4;
109
110      if(len & (AES_BLOCK_SIZE - 1))
111          return EXIT_FAILURE;
112
113  #if defined( USE_VIA_ACE_IF_PRESENT )
114
115      if(ctx->inf.b[1] == 0xff)
116      {   uint_8t *ksp = (uint_8t*)(ctx->ks);
117          via_cwd(cwd, hybrid, enc, 2 * ctx->inf.b[0] - 192);
118
119          if(ALIGN_OFFSET( ctx, 16 ))
120              return EXIT_FAILURE;
121
122          if(!ALIGN_OFFSET( ibuf, 16 ) && !ALIGN_OFFSET( obuf, 16 ))
123          {
124              via_ecb_op5(ksp, cwd, ibuf, obuf, nb);
125          }
126          else
127          {   aligned_auto(uint_8t, buf, BFR_BLOCKS * AES_BLOCK_SIZE,
↪16);

```

```

128         uint_8t *ip, *op;
129
130         while(nb)
131         {
132             int m = (nb > BFR_BLOCKS ? BFR_BLOCKS : nb);
133
134             ip = (ALIGN_OFFSET( ibuf, 16 ) ? buf : ibuf);
135             op = (ALIGN_OFFSET( obuf, 16 ) ? buf : obuf);
136
137             if(ip != ibuf)
138                 memcpy(buf, ibuf, m * AES_BLOCK_SIZE);
139
140             via_ecb_op5(ksp, cwd, ip, op, m);
141
142             if(op != obuf)
143                 memcpy(obuf, buf, m * AES_BLOCK_SIZE);
144
145             ibuf += m * AES_BLOCK_SIZE;
146             obuf += m * AES_BLOCK_SIZE;
147             nb -= m;
148         }
149     }
150
151     return EXIT_SUCCESS;
152 }
153
154 #endif
155
156 #if !defined( ASSUME_VIA_ACE_PRESENT )
157     while(nb--)
158     {
159         if(aes_encrypt(ibuf, obuf, ctx) != EXIT_SUCCESS)
160             return EXIT_FAILURE;
161         ibuf += AES_BLOCK_SIZE;
162         obuf += AES_BLOCK_SIZE;
163     }
164 #endif
165     return EXIT_SUCCESS;
166 }
167
168 AES_RETURN aes_ctr_pad(unsigned char *obuf, unsigned char *cbuf,
169 ↪ aes_encrypt_ctx ctx[1])
170 {
171     #if defined( USE_VIA_ACE_IF_PRESENT )
172         if(ctx->inf.b[1] == 0xff && ALIGN_OFFSET( ctx, 16 ))
173             return EXIT_FAILURE;
174     #endif
175     int i;
176     unsigned char acc;

```

```

176
177     memcpy(obuf, cbuf, AES_BLOCK_SIZE);
178     for (acc=1,i = AES_BLOCK_SIZE-1; i >= 0; i--) {
179         cbuf[i] += acc;
180         acc &= cbuf[i] == 0;
181     }
182     return aes_ecb_encrypt(obuf, obuf, AES_BLOCK_SIZE, ctx);
183 }
184
185 #if defined(__cplusplus)
186 }
187 #endif

```

src/Obf/aes_via_ace.h

```

1  /*
2  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
3  reserved.
4
5  The redistribution and use of this software (with or without changes)
6  is allowed without the payment of fees or royalties provided that:
7
8     source code distributions include the above copyright notice, this
9     list of conditions and the following disclaimer;
10
11    binary distributions include the above copyright notice, this list
12    of conditions and the following disclaimer in their documentation.
13
14    This software is provided 'as is' with no explicit or implied
15    warranties
16    in respect of its operation, including, but not limited to,
17    correctness
18    and fitness for purpose.
19
20    -----
21    Issue Date: 20/12/2007
22    */
23
24 #ifndef AES_VIA_ACE_H
25 #define AES_VIA_ACE_H
26
27 #if defined( _MSC_VER )
28 #   define INLINE __inline
29 #elif defined( __GNUC__ )
30 #   define INLINE static inline
31 #else
32 #   error VIA ACE requires Microsoft or GNU C
33 #endif
34
35 #define NEH_GENERATE 1

```

```

32 #define NEH_LOAD          2
33 #define NEH_HYBRID        3
34
35 #define MAX_READ_ATTEMPTS  1000
36
37 /* VIA Nehemiah RNG and ACE Feature Mask Values */
38
39 #define NEH_CPU_IS_VIA      0x00000001
40 #define NEH_CPU_READ        0x00000010
41 #define NEH_CPU_MASK        0x00000011
42
43 #define NEH_RNG_PRESENT     0x00000004
44 #define NEH_RNG_ENABLED     0x00000008
45 #define NEH_ACE_PRESENT     0x00000040
46 #define NEH_ACE_ENABLED     0x00000080
47 #define NEH_RNG_FLAGS       (NEH_RNG_PRESENT | NEH_RNG_ENABLED)
48 #define NEH_ACE_FLAGS       (NEH_ACE_PRESENT | NEH_ACE_ENABLED)
49 #define NEH_FLAGS_MASK      (NEH_RNG_FLAGS | NEH_ACE_FLAGS)
50
51 /* VIA Nehemiah Advanced Cryptography Engine (ACE) Control Word
   ↪ Values */
52
53 #define NEH_GEN_KEY          0x00000000    /* generate key schedule
   ↪ */
54 #define NEH_LOAD_KEY         0x00000080    /* load schedule from memory
   ↪ */
55 #define NEH_ENCRYPT           0x00000000    /* encryption
   ↪ */
56 #define NEH_DECRYPT           0x00000200    /* decryption
   ↪ */
57 #define NEH_KEY128           0x00000000+0x0a /* 128 bit key
   ↪ */
58 #define NEH_KEY192           0x00000400+0x0c /* 192 bit key
   ↪ */
59 #define NEH_KEY256           0x00000800+0x0e /* 256 bit key
   ↪ */
60
61 #define NEH_ENC_GEN           (NEH_ENCRYPT | NEH_GEN_KEY)
62 #define NEH_DEC_GEN           (NEH_DECRYPT | NEH_GEN_KEY)
63 #define NEH_ENC_LOAD         (NEH_ENCRYPT | NEH_LOAD_KEY)
64 #define NEH_DEC_LOAD         (NEH_DECRYPT | NEH_LOAD_KEY)
65
66 #define NEH_ENC_GEN_DATA {\
67     NEH_ENC_GEN | NEH_KEY128, 0, 0, 0,\
68     NEH_ENC_GEN | NEH_KEY192, 0, 0, 0,\
69     NEH_ENC_GEN | NEH_KEY256, 0, 0, 0 }
70
71 #define NEH_ENC_LOAD_DATA {\
72     NEH_ENC_LOAD | NEH_KEY128, 0, 0, 0,\

```

```

73     NEH_ENC_LOAD | NEH_KEY192, 0, 0, 0,\
74     NEH_ENC_LOAD | NEH_KEY256, 0, 0, 0 }
75
76 #define NEH_ENC_HYBRID_DATA {\
77     NEH_ENC_GEN   | NEH_KEY128, 0, 0, 0,\
78     NEH_ENC_LOAD  | NEH_KEY192, 0, 0, 0,\
79     NEH_ENC_LOAD  | NEH_KEY256, 0, 0, 0 }
80
81 #define NEH_DEC_GEN_DATA {\
82     NEH_DEC_GEN   | NEH_KEY128, 0, 0, 0,\
83     NEH_DEC_GEN   | NEH_KEY192, 0, 0, 0,\
84     NEH_DEC_GEN   | NEH_KEY256, 0, 0, 0 }
85
86 #define NEH_DEC_LOAD_DATA {\
87     NEH_DEC_LOAD  | NEH_KEY128, 0, 0, 0,\
88     NEH_DEC_LOAD  | NEH_KEY192, 0, 0, 0,\
89     NEH_DEC_LOAD  | NEH_KEY256, 0, 0, 0 }
90
91 #define NEH_DEC_HYBRID_DATA {\
92     NEH_DEC_GEN   | NEH_KEY128, 0, 0, 0,\
93     NEH_DEC_LOAD  | NEH_KEY192, 0, 0, 0,\
94     NEH_DEC_LOAD  | NEH_KEY256, 0, 0, 0 }
95
96 #define neh_enc_gen_key(x) ((x) == 128 ? (NEH_ENC_GEN | NEH_KEY128)
97 ↪: \
98     (x) == 192 ? (NEH_ENC_GEN | NEH_KEY192) : (NEH_ENC_GEN |
99 ↪NEH_KEY256))
100
101 #define neh_enc_load_key(x) ((x) == 128 ? (NEH_ENC_LOAD | NEH_KEY128)
102 ↪: \
103     (x) == 192 ? (NEH_ENC_LOAD | NEH_KEY192) : (NEH_ENC_LOAD |
104 ↪NEH_KEY256))
105
106 #define neh_enc_hybrid_key(x) ((x) == 128 ? (NEH_ENC_GEN |
107 ↪NEH_KEY128) : \
108     (x) == 192 ? (NEH_ENC_LOAD | NEH_KEY192) : (NEH_ENC_LOAD |
109 ↪NEH_KEY256))
110
111 #define neh_dec_gen_key(x) ((x) == 128 ? (NEH_DEC_GEN | NEH_KEY128)
112 ↪: \
113     (x) == 192 ? (NEH_DEC_GEN | NEH_KEY192) : (NEH_DEC_GEN |
114 ↪NEH_KEY256))
115
116 #define neh_dec_load_key(x) ((x) == 128 ? (NEH_DEC_LOAD | NEH_KEY128)
117 ↪: \
118     (x) == 192 ? (NEH_DEC_LOAD | NEH_KEY192) : (NEH_DEC_LOAD |
119 ↪NEH_KEY256))
120
121 #define neh_dec_hybrid_key(x) ((x) == 128 ? (NEH_DEC_GEN |
122 ↪NEH_KEY128) : \
123     (x) == 192 ? (NEH_DEC_LOAD | NEH_KEY192) : (NEH_DEC_LOAD |
124 ↪NEH_KEY256))

```

```

↪NEH_KEY128) : \
112     (x) == 192 ? (NEH_DEC_LOAD | NEH_KEY192) : (NEH_DEC_LOAD |
↪NEH_KEY256))
113
114 #if defined( _MSC_VER ) && ( _MSC_VER > 1200 )
115 #define aligned_auto(type, name, no, stride) __declspec(align(stride
↪)) type name[no]
116 #else
117 #define aligned_auto(type, name, no, stride) \
118     unsigned char _##name[no * sizeof(type) + stride]; \
119     type *name = (type*)(16 * (((unsigned long)(_##name)) + stride -
↪ 1) / stride))
120 #endif
121
122 #if defined( _MSC_VER ) && ( _MSC_VER > 1200 )
123 #define aligned_array(type, name, no, stride) __declspec(align(stride
↪)) type name[no]
124 #elif defined( __GNUC__ )
125 #define aligned_array(type, name, no, stride) type name[no]
↪__attribute__((aligned(stride)))
126 #else
127 #define aligned_array(type, name, no, stride) type name[no]
128 #endif
129
130 /* VIA ACE codeword */
131
132 static unsigned char via_flags = 0;
133
134 #if defined( _MSC_VER ) && ( _MSC_VER > 800 )
135
136 #define NEH_REKEY    __asm pushfd __asm popfd
137 #define NEH_AES      __asm _emit 0xf3 __asm _emit 0xf __asm _emit 0
↪0xa7
138 #define NEH_ECB      NEH_AES __asm _emit 0xc8
139 #define NEH_CBC      NEH_AES __asm _emit 0xd0
140 #define NEH_CFB      NEH_AES __asm _emit 0xe0
141 #define NEH_OFB      NEH_AES __asm _emit 0xe8
142 #define NEH_RNG      __asm _emit 0xf __asm _emit 0xa7 __asm _emit 0
↪0xc0
143
144 INLINE int has_cpuid(void)
145 {
146     char ret_value;
147     __asm
148     {
149         pushfd                /* save EFLAGS register */
150         mov     eax,[esp]     /* copy it to eax */
151         mov     edx,0x00200000 /* CPUID bit position */
152         xor     eax,edx       /* toggle the CPUID bit */
153         push    eax           /* attempt to set EFLAGS to */
154         popfd                /* the new value */

```

```

153     pushfd                /* get the new EFLAGS value */
154     pop     eax            /*      into eax          */
155     xor     eax,[esp]      /* xor with original value */
156     and     eax,edx        /* has CPUID bit changed? */
157     setne   al            /* set to 1 if we have been */
158     mov     ret_value,al   /*      able to change it   */
159     popfd                /* restore original EFLAGS */
160 }
161 return (int)ret_value;
162 }
163
164 inline int is_via_cpu(void)
165 {
166     __asm
167     {
168         push    ebx
169         xor     eax,eax      /* use CPUID to get vendor */
170         cpuid    /* identity string */
171         xor     eax,eax      /* is it "CentaurHauls" ? */
172         sub     ebx,0x746e6543 /* 'Cent' */
173         or      eax,ebx
174         sub     edx,0x48727561 /* 'aurH' */
175         or      eax,edx
176         sub     ecx,0x736c7561 /* 'auls' */
177         or      eax,ecx
178         sete    al          /* set to 1 if it is VIA ID */
179         mov     dl,NEH_CPU_READ /* mark CPU type as read */
180         or      dl,al        /* & store result in flags */
181         mov     [via_flags],dl /* set VIA detected flag */
182         mov     ret_value,al  /*      able to change it   */
183         pop     ebx
184     }
185     return (int)ret_value;
186 }
187
188 inline int read_via_flags(void)
189 {
190     __asm
191     {
192         mov     eax,0xC0000000 /* Centaur extended CPUID */
193         cpuid
194         mov     edx,0xC0000001 /* >= 0xC0000001 if support */
195         cmp     eax,edx        /* for VIA extended feature */
196         jnae    no_rng         /* flags is available */
197         mov     eax,edx        /* read Centaur extended */
198         cpuid    /* feature flags */
199         mov     eax,NEH_FLAGS_MASK /* mask out and save */
200         and     eax,edx        /* the RNG and ACE flags */
201         or      [via_flags],al /* present & enabled flags */
202         mov     ret_value,al   /*      able to change it   */
203     }
204     no_rng:

```



```

202     }
203     return (int)ret_value;
204 }
205
206 INLINE unsigned int via_rng_in(void *buf)
207 {
208     __asm
209     {
210         push    edi
211         mov     edi,buf          /* input buffer address */
212         xor     edx,edx          /* try to fetch 8 bytes */
213         NEH_RNG
214         and     ret_value,al     /* do RNG read operation */
215         pop     edi             /* count of bytes returned */
216     }
217     return (int)ret_value;
218 }
219
220 INLINE void via_ecb_op5(
221     const void *k, const void *c, const void *s, void *d, int
222     ↪ 1)
223 {
224     __asm
225     {
226         push    ebx
227         NEH_REKEY
228         mov     ebx, (k)
229         mov     edx, (c)
230         mov     esi, (s)
231         mov     edi, (d)
232         mov     ecx, (1)
233         NEH_ECB
234         pop     ebx
235     }
236 }
237
238 INLINE void via_cbc_op6(
239     const void *k, const void *c, const void *s, void *d, int
240     ↪ 1, void *v)
241 {
242     __asm
243     {
244         push    ebx
245         NEH_REKEY
246         mov     ebx, (k)
247         mov     edx, (c)
248         mov     esi, (s)
249         mov     edi, (d)
250         mov     ecx, (1)
251         mov     eax, (v)
252         NEH_CBC
253         pop     ebx
254     }
255 }

```

```

249
250 INLINE void via_cbc_op7(
251     const void *k, const void *c, const void *s, void *d, int l,
↪ void *v, void *w)
252 {
253     --asm
254     {
255         push     ebx
256         NEH_REKEY
257         mov     ebx, (k)
258         mov     edx, (c)
259         mov     esi, (s)
260         mov     edi, (d)
261         mov     ecx, (l)
262         mov     eax, (v)
263         NEH_CBC
264         mov     esi, eax
265         mov     edi, (w)
266         movsd
267         movsd
268         movsd
269         movsd
270         pop     ebx
271     }
272 }
273
274 INLINE void via_cfb_op6(
275     const void *k, const void *c, const void *s, void *d, int
↪ l, void *v)
276 {
277     --asm
278     {
279         push     ebx
280         NEH_REKEY
281         mov     ebx, (k)
282         mov     edx, (c)
283         mov     esi, (s)
284         mov     edi, (d)
285         mov     ecx, (l)
286         mov     eax, (v)
287         NEH_CFB
288         pop     ebx
289     }
290 }
291
292 INLINE void via_cfb_op7(
293     const void *k, const void *c, const void *s, void *d, int l,
↪ void *v, void *w)
294 {
295     --asm
296     {
297         push     ebx
298         NEH_REKEY
299         mov     ebx, (k)
300         mov     edx, (c)

```

```

295         mov     esi, (s)
296         mov     edi, (d)
297         mov     ecx, (l)
298         mov     eax, (v)
299         NEH_CFB
300         mov     esi, eax
301         mov     edi, (w)
302         movsd
303         movsd
304         movsd
305         movsd
306         pop     ebx
307     }
308 }
309
310 INLINE void via_ofb_op6(
311     const void *k, const void *c, const void *s, void *d, int
312     ↪ l, void *v)
313 {
314     __asm
315     {
316         push     ebx
317         NEH_REKEY
318         mov     ebx, (k)
319         mov     edx, (c)
320         mov     esi, (s)
321         mov     edi, (d)
322         mov     ecx, (l)
323         mov     eax, (v)
324         NEH_OFB
325         pop     ebx
326     }
327 }
328
329 #elif defined( __GNUC__ )
330
331 #define NEH_REKEY    asm("pushfl\n\tpopfl\n\t")
332 #define NEH_ECB      asm(".byte_0xf3,0x0f,0xa7,0xc8\n\t")
333 #define NEH_CBC       asm(".byte_0xf3,0x0f,0xa7,0xd0\n\t")
334 #define NEH_CFB       asm(".byte_0xf3,0x0f,0xa7,0xe0\n\t")
335 #define NEH_OFB       asm(".byte_0xf3,0x0f,0xa7,0xe8\n\t")
336 #define NEH_RNG       asm(".byte_0x0f,0xa7,0xc0\n\t");
337
338 INLINE int has_cpuid(void)
339 {
340     int val;
341     asm("pushfl\n\t");
342     asm("movl_0(%esp),%eax\n\t");
343     asm("xor_0x00200000,%eax\n\t");
344     asm("pushl_0(%esp)\n\t");
345     asm("popfl\n\t");
346     asm("pushfl\n\t");

```

```

343     asm("popl_%%eax\n\t");
344     asm("xorl_0(%%esp),%%edx\n\t");
345     asm("andl_0x00200000,%%eax\n\t");
346     asm("movl_%%eax,%0\n\t" : "=m" (val));
347     asm("popfl\n\t");
348     return val ? 1 : 0;
349 }
350
351 inline int is_via_cpu(void)
352 {
353     int val;
354     asm("pushl_%%ebx\n\t");
355     asm("xorl_%%eax,%%eax\n\t");
356     asm("cpuid\n\t");
357     asm("xorl_%%eax,%%eax\n\t");
358     asm("subl_0x746e6543,%%ebx\n\t");
359     asm("orl_%%ebx,%%eax\n\t");
360     asm("subl_0x48727561,%%edx\n\t");
361     asm("orl_%%edx,%%eax\n\t");
362     asm("subl_0x736c7561,%%ecx\n\t");
363     asm("orl_%%ecx,%%eax\n\t");
364     asm("movl_%%eax,%0\n\t" : "=m" (val));
365     val = (val ? 0 : 1);
366     via_flags = (val | NEH_CPU_READ);
367     return val;
368 }
369
370 inline int read_via_flags(void)
371 {
372     unsigned char val;
373     asm("movl_0xc0000000,%%eax\n\t");
374     asm("cpuid\n\t");
375     asm("movl_0xc0000001,%%edx\n\t");
376     asm("cmpl_%%edx,%%eax\n\t");
377     asm("setae_%%al\n\t");
378     asm("movb_%%al,%0\n\t" : "=m" (val));
379     if(!val) return 0;
380     asm("movl_0xc0000001,%%eax\n\t");
381     asm("cpuid\n\t");
382     asm("movb_%%dl,%0\n\t" : "=m" (val));
383     val &= NEH_FLAGS_MASK;
384     via_flags |= val;
385     return (int) val;
386 }
387
388 inline int via_rng_in(void *buf)
389 {
390     int val;
391     asm("pushl_%%edi\n\t");
392     asm("movl_0,%%edi\n\t" : : "m" (buf));
393     asm("xorl_%%edx,%%edx\n\t");

```

```

392     NEH_RNG
393     asm("andl_0x0000001f,%eax\n\t");
394     asm("movl_%%eax,%0\n\t" : "=m" (val));
395     asm("popl_%%edi\n\t");
396     return val;
397 }
398
399 INLINE volatile void via_ecb_op5(
400     const void *k, const void *c, const void *s, void *d, int
↪ 1)
401 {
402     asm("pushl_%%ebx\n\t");
403     NEH_REKEY;
404     asm("movl_0,%%ebx\n\t" : : "m" (k));
405     asm("movl_0,%%edx\n\t" : : "m" (c));
406     asm("movl_0,%%esi\n\t" : : "m" (s));
407     asm("movl_0,%%edi\n\t" : : "m" (d));
408     asm("movl_0,%%ecx\n\t" : : "m" (1));
409     NEH_ECB;
410     asm("popl_%%ebx\n\t");
411 }
412
413 INLINE volatile void via_cbc_op6(
414     const void *k, const void *c, const void *s, void *d, int
↪ 1, void *v)
415 {
416     asm("pushl_%%ebx\n\t");
417     NEH_REKEY;
418     asm("movl_0,%%ebx\n\t" : : "m" (k));
419     asm("movl_0,%%edx\n\t" : : "m" (c));
420     asm("movl_0,%%esi\n\t" : : "m" (s));
421     asm("movl_0,%%edi\n\t" : : "m" (d));
422     asm("movl_0,%%ecx\n\t" : : "m" (1));
423     asm("movl_0,%%eax\n\t" : : "m" (v));
424     NEH_CBC;
425     asm("popl_%%ebx\n\t");
426 }
427
428 INLINE volatile void via_cbc_op7(
429     const void *k, const void *c, const void *s, void *d, int l,
↪ void *v, void *w)
430 {
431     asm("pushl_%%ebx\n\t");
432     NEH_REKEY;
433     asm("movl_0,%%ebx\n\t" : : "m" (k));
434     asm("movl_0,%%edx\n\t" : : "m" (c));
435     asm("movl_0,%%esi\n\t" : : "m" (s));
436     asm("movl_0,%%edi\n\t" : : "m" (d));
437     asm("movl_0,%%ecx\n\t" : : "m" (l));

```

```

438     asm("movl_0,%%eax\n\t" : : "m" (v));
439     NEH_CBC;
440     asm("movl_0,%%esi\n\t");
441     asm("movl_0,%%edi\n\t" : : "m" (w));
442     asm("movsl;movsl;movsl;movsl\n\t");
443     asm("popl_0,%%ebx\n\t");
444 }
445
446 INLINE volatile void via_cfb_op6(
447     const void *k, const void *c, const void *s, void *d, int
↪ 1, void *v)
448 {
449     asm("pushl_0,%%ebx\n\t");
450     NEH_REKEY;
451     asm("movl_0,%%ebx\n\t" : : "m" (k));
452     asm("movl_0,%%edx\n\t" : : "m" (c));
453     asm("movl_0,%%esi\n\t" : : "m" (s));
454     asm("movl_0,%%edi\n\t" : : "m" (d));
455     asm("movl_0,%%ecx\n\t" : : "m" (1));
456     asm("movl_0,%%eax\n\t" : : "m" (v));
457     NEH_CFB;
458     asm("popl_0,%%ebx\n\t");
459 }
460
461 INLINE volatile void via_cfb_op7(
462     const void *k, const void *c, const void *s, void *d, int 1,
↪ void *v, void *w)
463 {
464     asm("pushl_0,%%ebx\n\t");
465     NEH_REKEY;
466     asm("movl_0,%%ebx\n\t" : : "m" (k));
467     asm("movl_0,%%edx\n\t" : : "m" (c));
468     asm("movl_0,%%esi\n\t" : : "m" (s));
469     asm("movl_0,%%edi\n\t" : : "m" (d));
470     asm("movl_0,%%ecx\n\t" : : "m" (1));
471     asm("movl_0,%%eax\n\t" : : "m" (v));
472     NEH_CFB;
473     asm("movl_0,%%esi\n\t");
474     asm("movl_0,%%edi\n\t" : : "m" (w));
475     asm("movsl;movsl;movsl;movsl\n\t");
476     asm("popl_0,%%ebx\n\t");
477 }
478
479 INLINE volatile void via_ofb_op6(
480     const void *k, const void *c, const void *s, void *d, int
↪ 1, void *v)
481 {
482     asm("pushl_0,%%ebx\n\t");
483     NEH_REKEY;

```

```

484     asm("movl%0,%ebx\n\t" : : "m" (k));
485     asm("movl%0,%edx\n\t" : : "m" (c));
486     asm("movl%0,%esi\n\t" : : "m" (s));
487     asm("movl%0,%edi\n\t" : : "m" (d));
488     asm("movl%0,%ecx\n\t" : : "m" (l));
489     asm("movl%0,%eax\n\t" : : "m" (v));
490     NEH_OFB;
491     asm("popl%ebx\n\t");
492 }
493
494 #else
495 #error VIA ACE is not available with this compiler
496 #endif
497
498 INLINE int via_ace_test(void)
499 {
500     return has_cpuid() && is_via_cpu() && ((read_via_flags() &
501     ↪NEH_ACE_FLAGS) == NEH_ACE_FLAGS);
502 }
503
504 #define VIA_ACE_AVAILABLE (((via_flags & NEH_ACE_FLAGS) ==
505     ↪NEH_ACE_FLAGS) \
506     || (via_flags & NEH_CPU_READ) && (via_flags & NEH_CPU_IS_VIA) ||
507     ↪via_ace_test())
508
509 INLINE int via_rng_test(void)
510 {
511     return has_cpuid() && is_via_cpu() && ((read_via_flags() &
512     ↪NEH_RNG_FLAGS) == NEH_RNG_FLAGS);
513 }
514
515 #define VIA_RNG_AVAILABLE (((via_flags & NEH_RNG_FLAGS) ==
516     ↪NEH_RNG_FLAGS) \
517     || (via_flags & NEH_CPU_READ) && (via_flags & NEH_CPU_IS_VIA) ||
518     ↪via_rng_test())
519
520 INLINE int read_via_rng(void *buf, int count)
521 {
522     int nbr, max_reads, lcnt = count;
523     unsigned char *p, *q;
524     aligned_auto(unsigned char, bp, 64, 16);
525
526     if(!VIA_RNG_AVAILABLE)
527         return 0;
528
529     do
530     {
531         max_reads = MAX_READ_ATTEMPTS;
532         do
533             nbr = via_rng_in(bp);

```

```

527         while
528             (nbr == 0 && --max_reads);
529
530         lcnt -= nbr;
531         p = (unsigned char*)buf; q = bp;
532         while(nbr--)
533             *p++ = *q++;
534     }
535     while
536         (lcnt && max_reads);
537
538     return count - lcnt;
539 }
540
541 #endif

```

src/Obf/aesencrypt.c

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24 */
25
26 #include "aesopt.h"
27 #include "aestab.h"
28
29 #if defined(__cplusplus)
30 extern "C"
31 {
32 #endif

```



```

28
29 #define si(y,x,k,c) (s(y,c) = word_in(x, c) ^ (k)[c])
30 #define so(y,x,c)   word_out(y, c, s(x,c))
31
32 #if defined(ARRAYS)
33 #define locals(y,x)   x[4],y[4]
34 #else
35 #define locals(y,x)   x##0,x##1,x##2,x##3,y##0,y##1,y##2,y##3
36 #endif
37
38 #define l_copy(y, x)   s(y,0) = s(x,0); s(y,1) = s(x,1); \
39                       s(y,2) = s(x,2); s(y,3) = s(x,3);
40 #define state_in(y,x,k) si(y,x,k,0); si(y,x,k,1); si(y,x,k,2); si(y,x
↪,k,3)
41 #define state_out(y,x) so(y,x,0); so(y,x,1); so(y,x,2); so(y,x,3)
42 #define round(rm,y,x,k) rm(y,x,k,0); rm(y,x,k,1); rm(y,x,k,2); rm(y,x
↪,k,3)
43
44 #if ( FUNCS_IN_C & ENCRYPTION_IN_C )
45
46 /* Visual C++ .Net v7.1 provides the fastest encryption code when
↪using
47     Pentium optimiation with small code but this is poor for
↪decryption
48     so we need to control this with the following VC++ pragmas
49 */
50
51 #if defined( _MSC_VER ) && !defined( _WIN64 )
52 #pragma optimize( "s", on )
53 #endif
54
55 /* Given the column (c) of the output state variable, the following
56     macros give the input state variables which are needed in its
57     computation for each row (r) of the state. All the alternative
58     macros give the same end values but expand into different ways
59     of calculating these values. In particular the complex macro
60     used for dynamically variable block sizes is designed to expand
61     to a compile time constant whenever possible but will expand to
62     conditional clauses on some branches (I am grateful to Frank
63     Yellin for this construction)
64 */
65
66 #define fwd_var(x,r,c)\
67     ( r == 0 ? ( c == 0 ? s(x,0) : c == 1 ? s(x,1) : c == 2 ? s(x,2) : s
↪(x,3))\
68     : r == 1 ? ( c == 0 ? s(x,1) : c == 1 ? s(x,2) : c == 2 ? s(x,3) : s
↪(x,0))\
69     : r == 2 ? ( c == 0 ? s(x,2) : c == 1 ? s(x,3) : c == 2 ? s(x,0) : s
↪(x,1))\

```

```

70 :      ( c == 0 ? s(x,3) : c == 1 ? s(x,0) : c == 2 ? s(x,1) : s
    ↪(x,2)))
71
72 #if defined(FT4_SET)
73 #undef dec_fmvars
74 #define fwd_rnd(y,x,k,c)      (s(y,c) = (k)[c] ^ four_tables(x,t_use(f,
    ↪n),fwd_var,rf1,c))
75 #elif defined(FT1_SET)
76 #undef dec_fmvars
77 #define fwd_rnd(y,x,k,c)      (s(y,c) = (k)[c] ^ one_table(x,upr,t_use(
    ↪f,n),fwd_var,rf1,c))
78 #else
79 #define fwd_rnd(y,x,k,c)      (s(y,c) = (k)[c] ^ fwd_mcol(no_table(x,
    ↪t_use(s,box),fwd_var,rf1,c)))
80 #endif
81
82 #if defined(FL4_SET)
83 #define fwd_lrnd(y,x,k,c)      (s(y,c) = (k)[c] ^ four_tables(x,t_use(f,
    ↪l),fwd_var,rf1,c))
84 #elif defined(FL1_SET)
85 #define fwd_lrnd(y,x,k,c)      (s(y,c) = (k)[c] ^ one_table(x,ups,t_use(
    ↪f,l),fwd_var,rf1,c))
86 #else
87 #define fwd_lrnd(y,x,k,c)      (s(y,c) = (k)[c] ^ no_table(x,t_use(s,box
    ↪),fwd_var,rf1,c))
88 #endif
89
90 AES_RETURN aes_encrypt(const unsigned char *in, unsigned char *out,
    ↪const aes_encrypt_ctx cx[1])
91 {   uint_32t      locals(b0, b1);
92     const uint_32t *kp;
93     #if defined( dec_fmvars )
94         dec_fmvars; /* declare variables for fwd_mcol() if needed */
95     #endif
96
97     if( cx->inf.b[0] != 10 * 16 && cx->inf.b[0] != 12 * 16 && cx->inf
    ↪.b[0] != 14 * 16 )
98         return EXIT_FAILURE;
99
100     kp = cx->ks;
101     state_in(b0, in, kp);
102
103     #if (ENC_UNROLL == FULL)
104
105     switch(cx->inf.b[0])
106     {
107     case 14 * 16:
108         round(fwd_rnd,  b1, b0, kp + 1 * N_COLS);
109         round(fwd_rnd,  b0, b1, kp + 2 * N_COLS);

```

```

110         kp += 2 * N_COLS;
111     case 12 * 16:
112         round(fwd_rnd, b1, b0, kp + 1 * N_COLS);
113         round(fwd_rnd, b0, b1, kp + 2 * N_COLS);
114         kp += 2 * N_COLS;
115     case 10 * 16:
116         round(fwd_rnd, b1, b0, kp + 1 * N_COLS);
117         round(fwd_rnd, b0, b1, kp + 2 * N_COLS);
118         round(fwd_rnd, b1, b0, kp + 3 * N_COLS);
119         round(fwd_rnd, b0, b1, kp + 4 * N_COLS);
120         round(fwd_rnd, b1, b0, kp + 5 * N_COLS);
121         round(fwd_rnd, b0, b1, kp + 6 * N_COLS);
122         round(fwd_rnd, b1, b0, kp + 7 * N_COLS);
123         round(fwd_rnd, b0, b1, kp + 8 * N_COLS);
124         round(fwd_rnd, b1, b0, kp + 9 * N_COLS);
125         round(fwd_lrnd, b0, b1, kp + 10 * N_COLS);
126     }
127
128     #else
129
130     #if (ENC_UNROLL == PARTIAL)
131     {
132         uint_32t rnd;
133         for(rnd = 0; rnd < (cx->inf.b[0] >> 5) - 1; ++rnd)
134         {
135             kp += N_COLS;
136             round(fwd_rnd, b1, b0, kp);
137             kp += N_COLS;
138             round(fwd_rnd, b0, b1, kp);
139         }
140         kp += N_COLS;
141         round(fwd_rnd, b1, b0, kp);
142     }
143     #else
144     {
145         uint_32t rnd;
146         for(rnd = 0; rnd < (cx->inf.b[0] >> 4) - 1; ++rnd)
147         {
148             kp += N_COLS;
149             round(fwd_rnd, b1, b0, kp);
150             l_copy(b0, b1);
151         }
152     }
153     #endif
154
155     kp += N_COLS;
156     round(fwd_lrnd, b0, b1, kp);
157 }
158
159     state_out(out, b0);
160     return EXIT_SUCCESS;
161 }

```

```

159 #endif
160
161 #if ( FUNCS_IN_C & DECRYPTION_IN_C)
162
163 /* Visual C++ .Net v7.1 provides the fastest encryption code when
164    ↪using
165    Pentium optimiation with small code but this is poor for
166    ↪decryption
167    so we need to control this with the following VC++ pragmas
168    */
169
170 #if defined( _MSC_VER ) && !defined( _WIN64 )
171 #pragma optimize( "t", on )
172 #endif
173
174 /* Given the column (c) of the output state variable, the following
175    macros give the input state variables which are needed in its
176    computation for each row (r) of the state. All the alternative
177    macros give the same end values but expand into different ways
178    of calculatating these values. In particular the complex macro
179    used for dynamically variable block sizes is designed to expand
180    to a compile time constant whenever possible but will expand to
181    conditional clauses on some branches (I am grateful to Frank
182    Yellin for this construction)
183    */
184
185 #define inv_var(x,r,c)\
186   ( r == 0 ? ( c == 0 ? s(x,0) : c == 1 ? s(x,1) : c == 2 ? s(x,2) : s
187   ↪(x,3))\
188   : r == 1 ? ( c == 0 ? s(x,3) : c == 1 ? s(x,0) : c == 2 ? s(x,1) : s
189   ↪(x,2))\
190   : r == 2 ? ( c == 0 ? s(x,2) : c == 1 ? s(x,3) : c == 2 ? s(x,0) : s
191   ↪(x,1))\
192   : ( c == 0 ? s(x,1) : c == 1 ? s(x,2) : c == 2 ? s(x,3) : s
193   ↪(x,0)))
194
195 #if defined(IT4_SET)
196 #undef dec_imvars
197 #define inv_rnd(y,x,k,c) (s(y,c) = (k)[c] ^ four_tables(x,t_use(i,
198 ↪n),inv_var,rf1,c))
199 #elif defined(IT1_SET)
200 #undef dec_imvars
201 #define inv_rnd(y,x,k,c) (s(y,c) = (k)[c] ^ one_table(x,upr,t_use(
202 ↪i,n),inv_var,rf1,c))
203 #else
204 #define inv_rnd(y,x,k,c) (s(y,c) = inv_mcol((k)[c] ^ no_table(x,
205 ↪t_use(i,box),inv_var,rf1,c)))
206 #endif
207
208

```

```

199 #if defined(IL4_SET)
200 #define inv_lrnd(y,x,k,c)    (s(y,c) = (k)[c] ^ four_tables(x,t_use(i,
    ↪l),inv_var,rf1,c))
201 #elif defined(IL1_SET)
202 #define inv_lrnd(y,x,k,c)    (s(y,c) = (k)[c] ^ one_table(x,ups,t_use(
    ↪i,l),inv_var,rf1,c))
203 #else
204 #define inv_lrnd(y,x,k,c)    (s(y,c) = (k)[c] ^ no_table(x,t_use(i,box
    ↪),inv_var,rf1,c))
205 #endif
206
207 /* This code can work with the decryption key schedule in the */
208 /* order that is used for encryption (where the 1st decryption */
209 /* round key is at the high end of the schedule) or with a key */
210 /* schedule that has been reversed to put the 1st decryption */
211 /* round key at the low end of the schedule in memory (when */
212 /* AES_REV_DKS is defined) */
213
214 #ifdef AES_REV_DKS
215 #define key_ofs    0
216 #define rnd_key(n) (kp + n * N_COLS)
217 #else
218 #define key_ofs    1
219 #define rnd_key(n) (kp - n * N_COLS)
220 #endif
221
222 AES_RETURN aes_decrypt(const unsigned char *in, unsigned char *out,
    ↪const aes_decrypt_ctx cx[1])
223 {
224     uint_32t    locals(b0, b1);
225     #if defined( dec_imvars )
226     dec_imvars; /* declare variables for inv_mcol() if needed */
227     #endif
228     const uint_32t *kp;
229
230     if( cx->inf.b[0] != 10 * 16 && cx->inf.b[0] != 12 * 16 && cx->inf
    ↪.b[0] != 14 * 16 )
231         return EXIT_FAILURE;
232
233     kp = cx->ks + (key_ofs ? (cx->inf.b[0] >> 2) : 0);
234     state_in(b0, in, kp);
235
236     #if (DEC_UNROLL == FULL)
237
238     kp = cx->ks + (key_ofs ? 0 : (cx->inf.b[0] >> 2));
239     switch(cx->inf.b[0])
240     {
241     case 14 * 16:
242         round(inv_rnd,  b1, b0, rnd_key(-13));
243         round(inv_rnd,  b0, b1, rnd_key(-12));

```

```

243     case 12 * 16:
244         round(inv_rnd, b1, b0, rnd_key(-11));
245         round(inv_rnd, b0, b1, rnd_key(-10));
246     case 10 * 16:
247         round(inv_rnd, b1, b0, rnd_key(-9));
248         round(inv_rnd, b0, b1, rnd_key(-8));
249         round(inv_rnd, b1, b0, rnd_key(-7));
250         round(inv_rnd, b0, b1, rnd_key(-6));
251         round(inv_rnd, b1, b0, rnd_key(-5));
252         round(inv_rnd, b0, b1, rnd_key(-4));
253         round(inv_rnd, b1, b0, rnd_key(-3));
254         round(inv_rnd, b0, b1, rnd_key(-2));
255         round(inv_rnd, b1, b0, rnd_key(-1));
256         round(inv_lrnd, b0, b1, rnd_key( 0));
257     }
258
259 #else
260
261 #if (DEC_UNROLL == PARTIAL)
262     {
263         uint_32t    rnd;
264         for(rnd = 0; rnd < (cx->inf.b[0] >> 5) - 1; ++rnd)
265         {
266             kp = rnd_key(1);
267             round(inv_rnd, b1, b0, kp);
268             kp = rnd_key(1);
269             round(inv_rnd, b0, b1, kp);
270         }
271         kp = rnd_key(1);
272         round(inv_rnd, b1, b0, kp);
273     }
274 #else
275     {
276         uint_32t    rnd;
277         for(rnd = 0; rnd < (cx->inf.b[0] >> 4) - 1; ++rnd)
278         {
279             kp = rnd_key(1);
280             round(inv_rnd, b1, b0, kp);
281             l_copy(b0, b1);
282         }
283     }
284 #endif
285
286     kp = rnd_key(1);
287     round(inv_lrnd, b0, b1, kp);
288 }
289
290 #endif
291

```

```

292 #if defined(__cplusplus)
293 }
294 #endif

src/Obf/aeskey.c

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24 */
25
26 #include "aesopt.h"
27 #include "aestab.h"
28
29 #ifdef USE_VIA_ACE_IF_PRESENT
30 # include "aes_via_ace.h"
31 #endif
32
33 #if defined(__cplusplus)
34 extern "C"
35 {
36 #endif
37
38 /* Initialise the key schedule from the user supplied key. The key
39 length can be specified in bytes, with legal values of 16, 24
40 and 32, or in bits, with legal values of 128, 192 and 256. These
41 values correspond with Nk values of 4, 6 and 8 respectively.
42
43 The following macros implement a single cycle in the key
44 schedule generation process. The number of cycles needed

```

```

40     for each cx->n_col and nk value is:
41
42     nk =           4  5  6  7  8
43     -----
44     cx->n_col = 4    10  9  8  7  7
45     cx->n_col = 5    14 11 10  9  9
46     cx->n_col = 6    19 15 12 11 11
47     cx->n_col = 7    21 19 16 13 14
48     cx->n_col = 8    29 23 19 17 14
49 */
50
51 #if defined( REDUCE_CODE_SIZE )
52 #   define ls_box ls_sub
53     uint_32t ls_sub(const uint_32t t, const uint_32t n);
54 #   define inv_mcol im_sub
55     uint_32t im_sub(const uint_32t x);
56 #   ifdef ENC_KS_UNROLL
57 #       undef ENC_KS_UNROLL
58 #   endif
59 #   ifdef DEC_KS_UNROLL
60 #       undef DEC_KS_UNROLL
61 #   endif
62 #endif
63
64 #if (FUNCS_IN_C & ENC_KEYING_IN_C)
65
66 #if defined(AES_128) || defined( AES_VAR )
67
68 #define ke4(k,i) \
69 {   k[4*(i)+4] = ss[0] ^= ls_box(ss[3],3) ^ t_use(r,c)[i]; \
70     k[4*(i)+5] = ss[1] ^= ss[0]; \
71     k[4*(i)+6] = ss[2] ^= ss[1]; \
72     k[4*(i)+7] = ss[3] ^= ss[2]; \
73 }
74
75 AES_RETURN aes_encrypt_key128(const unsigned char *key,
76                               ↪aes_encrypt_ctx cx[1])
77 {   uint_32t    ss[4];
78
79     cx->ks[0] = ss[0] = word_in(key, 0);
80     cx->ks[1] = ss[1] = word_in(key, 1);
81     cx->ks[2] = ss[2] = word_in(key, 2);
82     cx->ks[3] = ss[3] = word_in(key, 3);
83
84 #ifdef ENC_KS_UNROLL
85     ke4(cx->ks, 0);  ke4(cx->ks, 1);
86     ke4(cx->ks, 2);  ke4(cx->ks, 3);
87     ke4(cx->ks, 4);  ke4(cx->ks, 5);
88     ke4(cx->ks, 6);  ke4(cx->ks, 7);

```



```

88     ke4(cx->ks, 8);
89 #else
90     {    uint_32t i;
91         for(i = 0; i < 9; ++i)
92             ke4(cx->ks, i);
93     }
94 #endif
95     ke4(cx->ks, 9);
96     cx->inf.l = 0;
97     cx->inf.b[0] = 10 * 16;
98
99 #ifdef USE_VIA_ACE_IF_PRESENT
100     if(VIA_ACE_AVAILABLE)
101         cx->inf.b[1] = 0xff;
102 #endif
103     return EXIT_SUCCESS;
104 }
105
106 #endif
107
108 #if defined(AES_192) || defined( AES_VAR )
109
110 #define kef6(k,i) \
111 {    k[6*(i)+ 6] = ss[0] ^= ls_box(ss[5],3) ^ t_use(r,c)[i]; \
112     k[6*(i)+ 7] = ss[1] ^= ss[0]; \
113     k[6*(i)+ 8] = ss[2] ^= ss[1]; \
114     k[6*(i)+ 9] = ss[3] ^= ss[2]; \
115 }
116
117 #define ke6(k,i) \
118 {    kef6(k,i); \
119     k[6*(i)+10] = ss[4] ^= ss[3]; \
120     k[6*(i)+11] = ss[5] ^= ss[4]; \
121 }
122
123 AES_RETURN aes_encrypt_key192(const unsigned char *key,
124 ↪aes_encrypt_ctx cx[1])
125 {    uint_32t    ss[6];
126
127     cx->ks[0] = ss[0] = word_in(key, 0);
128     cx->ks[1] = ss[1] = word_in(key, 1);
129     cx->ks[2] = ss[2] = word_in(key, 2);
130     cx->ks[3] = ss[3] = word_in(key, 3);
131     cx->ks[4] = ss[4] = word_in(key, 4);
132     cx->ks[5] = ss[5] = word_in(key, 5);
133
134 #ifdef ENC_KS_UNROLL
135     ke6(cx->ks, 0);    ke6(cx->ks, 1);
136     ke6(cx->ks, 2);    ke6(cx->ks, 3);

```

```

136     ke6(cx->ks, 4);  ke6(cx->ks, 5);
137     ke6(cx->ks, 6);
138 #else
139     {   uint_32t i;
140         for(i = 0; i < 7; ++i)
141             ke6(cx->ks, i);
142     }
143 #endif
144     kef6(cx->ks, 7);
145     cx->inf.l = 0;
146     cx->inf.b[0] = 12 * 16;
147
148 #ifdef USE_VIA_ACE_IF_PRESENT
149     if(VIA_ACE_AVAILABLE)
150         cx->inf.b[1] = 0xff;
151 #endif
152     return EXIT_SUCCESS;
153 }
154
155 #endif
156
157 #if defined(AES_256) || defined( AES_VAR )
158
159 #define kef8(k,i) \
160 {   k[8*(i)+ 8] = ss[0] ^= ls_box(ss[7],3) ^ t_use(r,c)[i]; \
161     k[8*(i)+ 9] = ss[1] ^= ss[0]; \
162     k[8*(i)+10] = ss[2] ^= ss[1]; \
163     k[8*(i)+11] = ss[3] ^= ss[2]; \
164 }
165
166 #define ke8(k,i) \
167 {   kef8(k,i); \
168     k[8*(i)+12] = ss[4] ^= ls_box(ss[3],0); \
169     k[8*(i)+13] = ss[5] ^= ss[4]; \
170     k[8*(i)+14] = ss[6] ^= ss[5]; \
171     k[8*(i)+15] = ss[7] ^= ss[6]; \
172 }
173
174 AES_RETURN aes_encrypt_key256(const unsigned char *key,
175 ↪aes_encrypt_ctx cx[1])
176 {   uint_32t    ss[8];
177
178     cx->ks[0] = ss[0] = word_in(key, 0);
179     cx->ks[1] = ss[1] = word_in(key, 1);
180     cx->ks[2] = ss[2] = word_in(key, 2);
181     cx->ks[3] = ss[3] = word_in(key, 3);
182     cx->ks[4] = ss[4] = word_in(key, 4);
183     cx->ks[5] = ss[5] = word_in(key, 5);
184     cx->ks[6] = ss[6] = word_in(key, 6);

```

```

184     cx->ks[7] = ss[7] = word_in(key, 7);
185
186 #ifdef ENC_KS_UNROLL
187     ke8(cx->ks, 0); ke8(cx->ks, 1);
188     ke8(cx->ks, 2); ke8(cx->ks, 3);
189     ke8(cx->ks, 4); ke8(cx->ks, 5);
190 #else
191     {   uint_32t i;
192         for(i = 0; i < 6; ++i)
193             ke8(cx->ks, i);
194     }
195 #endif
196     kef8(cx->ks, 6);
197     cx->inf.l = 0;
198     cx->inf.b[0] = 14 * 16;
199
200 #ifdef USE_VIA_ACE_IF_PRESENT
201     if(VIA_ACE_AVAILABLE)
202         cx->inf.b[1] = 0xff;
203 #endif
204     return EXIT_SUCCESS;
205 }
206
207 #endif
208
209 #if defined( AES_VAR )
210
211 AES_RETURN aes_encrypt_key(const unsigned char *key, int key_len,
212 ↪aes_encrypt_ctx cx[1])
213 {
214     switch(key_len)
215     {
216     case 16: case 128: return aes_encrypt_key128(key, cx);
217     case 24: case 192: return aes_encrypt_key192(key, cx);
218     case 32: case 256: return aes_encrypt_key256(key, cx);
219     default: return EXIT_FAILURE;
220     }
221 }
222 #endif
223
224 #endif
225
226 #if (FUNCS_IN_C & DEC_KEYING_IN_C)
227
228 /* this is used to store the decryption round keys */
229 /* in forward or reverse order */
230
231 #ifdef AES_REV_DKS

```

```

232 #define v(n,i) ((n) - (i) + 2 * ((i) & 3))
233 #else
234 #define v(n,i) (i)
235 #endif
236
237 #if DEC_ROUND == NO_TABLES
238 #define ff(x) (x)
239 #else
240 #define ff(x) inv_mcol(x)
241 #if defined( dec_imvars )
242 #define d_vars dec_imvars
243 #endif
244 #endif
245
246 #if defined(AES_128) || defined( AES_VAR )
247
248 #define k4e(k,i) \
249 {   k[v(40,(4*(i))+4)] = ss[0] ^= ls_box(ss[3],3) ^ t_use(r,c)[i]; \
250     k[v(40,(4*(i))+5)] = ss[1] ^= ss[0]; \
251     k[v(40,(4*(i))+6)] = ss[2] ^= ss[1]; \
252     k[v(40,(4*(i))+7)] = ss[3] ^= ss[2]; \
253 }
254
255 #if 1
256
257 #define kdf4(k,i) \
258 {   ss[0] = ss[0] ^ ss[2] ^ ss[1] ^ ss[3]; \
259     ss[1] = ss[1] ^ ss[3]; \
260     ss[2] = ss[2] ^ ss[3]; \
261     ss[4] = ls_box(ss[(i+3) % 4], 3) ^ t_use(r,c)[i]; \
262     ss[i % 4] ^= ss[4]; \
263     ss[4] ^= k[v(40,(4*(i)))]; k[v(40,(4*(i))+4)] = ff(ss[4]); \
264     ss[4] ^= k[v(40,(4*(i))+1)]; k[v(40,(4*(i))+5)] = ff(ss[4]); \
265     ss[4] ^= k[v(40,(4*(i))+2)]; k[v(40,(4*(i))+6)] = ff(ss[4]); \
266     ss[4] ^= k[v(40,(4*(i))+3)]; k[v(40,(4*(i))+7)] = ff(ss[4]); \
267 }
268
269 #define kd4(k,i) \
270 {   ss[4] = ls_box(ss[(i+3) % 4], 3) ^ t_use(r,c)[i]; \
271     ss[i % 4] ^= ss[4]; ss[4] = ff(ss[4]); \
272     k[v(40,(4*(i))+4)] = ss[4] ^= k[v(40,(4*(i)))]; \
273     k[v(40,(4*(i))+5)] = ss[4] ^= k[v(40,(4*(i))+1)]; \
274     k[v(40,(4*(i))+6)] = ss[4] ^= k[v(40,(4*(i))+2)]; \
275     k[v(40,(4*(i))+7)] = ss[4] ^= k[v(40,(4*(i))+3)]; \
276 }
277
278 #define kdl4(k,i) \
279 {   ss[4] = ls_box(ss[(i+3) % 4], 3) ^ t_use(r,c)[i]; ss[i % 4] ^= ss
↪[4]; \

```

```

280     k[v(40,(4*(i))+4)] = (ss[0] ^= ss[1]) ^ ss[2] ^ ss[3]; \
281     k[v(40,(4*(i))+5)] = ss[1] ^ ss[3]; \
282     k[v(40,(4*(i))+6)] = ss[0]; \
283     k[v(40,(4*(i))+7)] = ss[1]; \
284 }
285
286 #else
287
288 #define kdf4(k,i) \
289 {   ss[0] ^= ls_box(ss[3],3) ^ t_use(r,c)[i]; k[v(40,(4*(i))+ 4)] =
    ↪ff(ss[0]); \
290     ss[1] ^= ss[0]; k[v(40,(4*(i))+ 5)] = ff(ss[1]); \
291     ss[2] ^= ss[1]; k[v(40,(4*(i))+ 6)] = ff(ss[2]); \
292     ss[3] ^= ss[2]; k[v(40,(4*(i))+ 7)] = ff(ss[3]); \
293 }
294
295 #define kd4(k,i) \
296 {   ss[4] = ls_box(ss[3],3) ^ t_use(r,c)[i]; \
297     ss[0] ^= ss[4]; ss[4] = ff(ss[4]); k[v(40,(4*(i))+ 4)] = ss[4] ^=
    ↪k[v(40,(4*(i)))]; \
298     ss[1] ^= ss[0]; k[v(40,(4*(i))+ 5)] = ss[4] ^= k[v(40,(4*(i))+ 1)
    ↪]; \
299     ss[2] ^= ss[1]; k[v(40,(4*(i))+ 6)] = ss[4] ^= k[v(40,(4*(i))+ 2)
    ↪]; \
300     ss[3] ^= ss[2]; k[v(40,(4*(i))+ 7)] = ss[4] ^= k[v(40,(4*(i))+ 3)
    ↪]; \
301 }
302
303 #define kdl4(k,i) \
304 {   ss[0] ^= ls_box(ss[3],3) ^ t_use(r,c)[i]; k[v(40,(4*(i))+ 4)] =
    ↪ss[0]; \
305     ss[1] ^= ss[0]; k[v(40,(4*(i))+ 5)] = ss[1]; \
306     ss[2] ^= ss[1]; k[v(40,(4*(i))+ 6)] = ss[2]; \
307     ss[3] ^= ss[2]; k[v(40,(4*(i))+ 7)] = ss[3]; \
308 }
309
310 #endif
311
312 AES_RETURN aes_decrypt_key128(const unsigned char *key,
    ↪aes_decrypt_ctx cx[1])
313 {   uint_32t    ss[5];
314     #if defined( d_vars )
315         d_vars;
316     #endif
317     cx->ks[v(40,(0))] = ss[0] = word_in(key, 0);
318     cx->ks[v(40,(1))] = ss[1] = word_in(key, 1);
319     cx->ks[v(40,(2))] = ss[2] = word_in(key, 2);
320     cx->ks[v(40,(3))] = ss[3] = word_in(key, 3);
321

```

```

322 #ifdef DEC_KS_UNROLL
323     kdf4(cx->ks, 0); kd4(cx->ks, 1);
324     kd4(cx->ks, 2); kd4(cx->ks, 3);
325     kd4(cx->ks, 4); kd4(cx->ks, 5);
326     kd4(cx->ks, 6); kd4(cx->ks, 7);
327     kd4(cx->ks, 8); kd14(cx->ks, 9);
328 #else
329     { uint_32t i;
330       for(i = 0; i < 10; ++i)
331         k4e(cx->ks, i);
332     #if !(DEC_ROUND == NO_TABLES)
333       for(i = N_COLS; i < 10 * N_COLS; ++i)
334         cx->ks[i] = inv_mcol(cx->ks[i]);
335     #endif
336     }
337 #endif
338     cx->inf.l = 0;
339     cx->inf.b[0] = 10 * 16;
340
341 #ifdef USE_VIA_ACE_IF_PRESENT
342     if(VIA_ACE_AVAILABLE)
343         cx->inf.b[1] = 0xff;
344 #endif
345     return EXIT_SUCCESS;
346 }
347
348 #endif
349
350 #if defined(AES_192) || defined( AES_VAR )
351
352 #define k6ef(k,i) \
353 { k[v(48,(6*(i))+ 6)] = ss[0] ^= ls_box(ss[5],3) ^ t_use(r,c)[i]; \
354   k[v(48,(6*(i))+ 7)] = ss[1] ^= ss[0]; \
355   k[v(48,(6*(i))+ 8)] = ss[2] ^= ss[1]; \
356   k[v(48,(6*(i))+ 9)] = ss[3] ^= ss[2]; \
357 }
358
359 #define k6e(k,i) \
360 { k6ef(k,i); \
361   k[v(48,(6*(i))+10)] = ss[4] ^= ss[3]; \
362   k[v(48,(6*(i))+11)] = ss[5] ^= ss[4]; \
363 }
364
365 #define kdf6(k,i) \
366 { ss[0] ^= ls_box(ss[5],3) ^ t_use(r,c)[i]; k[v(48,(6*(i))+ 6)] = \
367   ~ff(ss[0]); \
368   ss[1] ^= ss[0]; k[v(48,(6*(i))+ 7)] = ff(ss[1]); \
369   ss[2] ^= ss[1]; k[v(48,(6*(i))+ 8)] = ff(ss[2]); \
370   ss[3] ^= ss[2]; k[v(48,(6*(i))+ 9)] = ff(ss[3]); \

```

```

370     ss[4] ^= ss[3]; k[v(48,(6*(i))+10)] = ff(ss[4]); \
371     ss[5] ^= ss[4]; k[v(48,(6*(i))+11)] = ff(ss[5]); \
372 }
373
374 #define kd6(k,i) \
375 {   ss[6] = ls_box(ss[5],3) ^ t_use(r,c)[i]; \
376     ss[0] ^= ss[6]; ss[6] = ff(ss[6]); k[v(48,(6*(i))+ 6)] = ss[6] ^=
↪ k[v(48,(6*(i)))]; \
377     ss[1] ^= ss[0]; k[v(48,(6*(i))+ 7)] = ss[6] ^= k[v(48,(6*(i))+ 1)
↪]; \
378     ss[2] ^= ss[1]; k[v(48,(6*(i))+ 8)] = ss[6] ^= k[v(48,(6*(i))+ 2)
↪]; \
379     ss[3] ^= ss[2]; k[v(48,(6*(i))+ 9)] = ss[6] ^= k[v(48,(6*(i))+ 3)
↪]; \
380     ss[4] ^= ss[3]; k[v(48,(6*(i))+10)] = ss[6] ^= k[v(48,(6*(i))+ 4)
↪]; \
381     ss[5] ^= ss[4]; k[v(48,(6*(i))+11)] = ss[6] ^= k[v(48,(6*(i))+ 5)
↪]; \
382 }
383
384 #define kdl6(k,i) \
385 {   ss[0] ^= ls_box(ss[5],3) ^ t_use(r,c)[i]; k[v(48,(6*(i))+ 6)] =
↪ss[0]; \
386     ss[1] ^= ss[0]; k[v(48,(6*(i))+ 7)] = ss[1]; \
387     ss[2] ^= ss[1]; k[v(48,(6*(i))+ 8)] = ss[2]; \
388     ss[3] ^= ss[2]; k[v(48,(6*(i))+ 9)] = ss[3]; \
389 }
390
391 AES_RETURN aes_decrypt_key192(const unsigned char *key,
↪aes_decrypt_ctx cx[1])
392 {   uint_32t    ss[7];
393     #if defined( d_vars )
394         d_vars;
395     #endif
396     cx->ks[v(48,(0))] = ss[0] = word_in(key, 0);
397     cx->ks[v(48,(1))] = ss[1] = word_in(key, 1);
398     cx->ks[v(48,(2))] = ss[2] = word_in(key, 2);
399     cx->ks[v(48,(3))] = ss[3] = word_in(key, 3);
400
401     #ifdef DEC_KS_UNROLL
402         cx->ks[v(48,(4))] = ff(ss[4] = word_in(key, 4));
403         cx->ks[v(48,(5))] = ff(ss[5] = word_in(key, 5));
404         kdf6(cx->ks, 0); kd6(cx->ks, 1);
405         kd6(cx->ks, 2);  kd6(cx->ks, 3);
406         kd6(cx->ks, 4);  kd6(cx->ks, 5);
407         kd6(cx->ks, 6);  kdl6(cx->ks, 7);
408     #else
409         cx->ks[v(48,(4))] = ss[4] = word_in(key, 4);
410         cx->ks[v(48,(5))] = ss[5] = word_in(key, 5);

```

```

411     {    uint_32t i;
412
413         for(i = 0; i < 7; ++i)
414             k6e(cx->ks, i);
415         k6ef(cx->ks, 7);
416 #if !(DEC_ROUND == NO_TABLES)
417         for(i = N_COLS; i < 12 * N_COLS; ++i)
418             cx->ks[i] = inv_mcol(cx->ks[i]);
419 #endif
420     }
421 #endif
422     cx->inf.l = 0;
423     cx->inf.b[0] = 12 * 16;
424
425 #ifdef USE_VIA_ACE_IF_PRESENT
426     if(VIA_ACE_AVAILABLE)
427         cx->inf.b[1] = 0xff;
428 #endif
429     return EXIT_SUCCESS;
430 }
431
432 #endif
433
434 #if defined(AES_256) || defined( AES_VAR )
435
436 #define k8ef(k,i) \
437 {    k[v(56,(8*(i))+ 8)] = ss[0] ^= ls_box(ss[7],3) ^ t_use(r,c)[i]; \
438     k[v(56,(8*(i))+ 9)] = ss[1] ^= ss[0]; \
439     k[v(56,(8*(i))+10)] = ss[2] ^= ss[1]; \
440     k[v(56,(8*(i))+11)] = ss[3] ^= ss[2]; \
441 }
442
443 #define k8e(k,i) \
444 {    k8ef(k,i); \
445     k[v(56,(8*(i))+12)] = ss[4] ^= ls_box(ss[3],0); \
446     k[v(56,(8*(i))+13)] = ss[5] ^= ss[4]; \
447     k[v(56,(8*(i))+14)] = ss[6] ^= ss[5]; \
448     k[v(56,(8*(i))+15)] = ss[7] ^= ss[6]; \
449 }
450
451 #define kdf8(k,i) \
452 {    ss[0] ^= ls_box(ss[7],3) ^ t_use(r,c)[i]; k[v(56,(8*(i))+ 8)] =
453     ↪ff(ss[0]); \
454     ss[1] ^= ss[0]; k[v(56,(8*(i))+ 9)] = ff(ss[1]); \
455     ss[2] ^= ss[1]; k[v(56,(8*(i))+10)] = ff(ss[2]); \
456     ss[3] ^= ss[2]; k[v(56,(8*(i))+11)] = ff(ss[3]); \
457     ss[4] ^= ls_box(ss[3],0); k[v(56,(8*(i))+12)] = ff(ss[4]); \
458     ss[5] ^= ss[4]; k[v(56,(8*(i))+13)] = ff(ss[5]); \
459     ss[6] ^= ss[5]; k[v(56,(8*(i))+14)] = ff(ss[6]); \

```



```

459     ss[7] ^= ss[6]; k[v(56,(8*(i))+15)] = ff(ss[7]); \
460 }
461
462 #define kd8(k,i) \
463 {   ss[8] = ls_box(ss[7],3) ^ t_use(r,c)[i]; \
464     ss[0] ^= ss[8]; ss[8] = ff(ss[8]); k[v(56,(8*(i))+ 8)] = ss[8] ^=
↪ k[v(56,(8*(i)))]; \
465     ss[1] ^= ss[0]; k[v(56,(8*(i))+ 9)] = ss[8] ^= k[v(56,(8*(i))+ 1)
↪]; \
466     ss[2] ^= ss[1]; k[v(56,(8*(i))+10)] = ss[8] ^= k[v(56,(8*(i))+ 2)
↪]; \
467     ss[3] ^= ss[2]; k[v(56,(8*(i))+11)] = ss[8] ^= k[v(56,(8*(i))+ 3)
↪]; \
468     ss[8] = ls_box(ss[3],0); \
469     ss[4] ^= ss[8]; ss[8] = ff(ss[8]); k[v(56,(8*(i))+12)] = ss[8] ^=
↪ k[v(56,(8*(i))+ 4)]; \
470     ss[5] ^= ss[4]; k[v(56,(8*(i))+13)] = ss[8] ^= k[v(56,(8*(i))+ 5)
↪]; \
471     ss[6] ^= ss[5]; k[v(56,(8*(i))+14)] = ss[8] ^= k[v(56,(8*(i))+ 6)
↪]; \
472     ss[7] ^= ss[6]; k[v(56,(8*(i))+15)] = ss[8] ^= k[v(56,(8*(i))+ 7)
↪]; \
473 }
474
475 #define kdl8(k,i) \
476 {   ss[0] ^= ls_box(ss[7],3) ^ t_use(r,c)[i]; k[v(56,(8*(i))+ 8)] =
↪ss[0]; \
477     ss[1] ^= ss[0]; k[v(56,(8*(i))+ 9)] = ss[1]; \
478     ss[2] ^= ss[1]; k[v(56,(8*(i))+10)] = ss[2]; \
479     ss[3] ^= ss[2]; k[v(56,(8*(i))+11)] = ss[3]; \
480 }
481
482 AES_RETURN aes_decrypt_key256(const unsigned char *key,
↪aes_decrypt_ctx cx[1])
483 {   uint_32t    ss[9];
484     #if defined( d_vars )
485         d_vars;
486     #endif
487     cx->ks[v(56,(0))] = ss[0] = word_in(key, 0);
488     cx->ks[v(56,(1))] = ss[1] = word_in(key, 1);
489     cx->ks[v(56,(2))] = ss[2] = word_in(key, 2);
490     cx->ks[v(56,(3))] = ss[3] = word_in(key, 3);
491
492     #ifdef DEC_KS_UNROLL
493     cx->ks[v(56,(4))] = ff(ss[4] = word_in(key, 4));
494     cx->ks[v(56,(5))] = ff(ss[5] = word_in(key, 5));
495     cx->ks[v(56,(6))] = ff(ss[6] = word_in(key, 6));
496     cx->ks[v(56,(7))] = ff(ss[7] = word_in(key, 7));
497     kdf8(cx->ks, 0); kd8(cx->ks, 1);

```

```

498     kd8(cx->ks, 2);  kd8(cx->ks, 3);
499     kd8(cx->ks, 4);  kd8(cx->ks, 5);
500     kdl8(cx->ks, 6);
501 #else
502     cx->ks[v(56,(4))] = ss[4] = word_in(key, 4);
503     cx->ks[v(56,(5))] = ss[5] = word_in(key, 5);
504     cx->ks[v(56,(6))] = ss[6] = word_in(key, 6);
505     cx->ks[v(56,(7))] = ss[7] = word_in(key, 7);
506     {    uint_32t i;
507
508         for(i = 0; i < 6; ++i)
509             k8e(cx->ks, i);
510         k8ef(cx->ks, 6);
511 #if !(DEC_ROUND == NO_TABLES)
512         for(i = N_COLS; i < 14 * N_COLS; ++i)
513             cx->ks[i] = inv_mcol(cx->ks[i]);
514 #endif
515     }
516 #endif
517     cx->inf.l = 0;
518     cx->inf.b[0] = 14 * 16;
519
520 #ifdef USE_VIA_ACE_IF_PRESENT
521     if(VIA_ACE_AVAILABLE)
522         cx->inf.b[1] = 0xff;
523 #endif
524     return EXIT_SUCCESS;
525 }
526
527 #endif
528
529 #if defined( AES_VAR )
530
531 AES_RETURN aes_decrypt_key(const unsigned char *key, int key_len,
532 ↪aes_decrypt_ctx cx[1])
533 {
534     switch(key_len)
535     {
536     case 16: case 128: return aes_decrypt_key128(key, cx);
537     case 24: case 192: return aes_decrypt_key192(key, cx);
538     case 32: case 256: return aes_decrypt_key256(key, cx);
539     default: return EXIT_FAILURE;
540     }
541 }
542 #endif
543
544 #endif
545

```

```

546 #if defined(__cplusplus)
547 }
548 #endif

```

src/Obf/aesopt.h

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24
25 This file contains the compilation options for AES (Rijndael) and
26 ↪code
27 that is common across encryption, key scheduling and table
28 ↪generation.
29
30 OPERATION
31
32 These source code files implement the AES algorithm Rijndael
33 ↪designed by
34 Joan Daemen and Vincent Rijmen. This version is designed for the
35 ↪standard
36 block size of 16 bytes and for key sizes of 128, 192 and 256 bits
37 ↪(16, 24
38 and 32 bytes).
39
40 This version is designed for flexibility and speed using operations
41 ↪on
42 32-bit words rather than operations on bytes. It can be compiled
43 ↪with
44 either big or little endian internal byte order but is faster when

```

```

33     ↪the
34     native byte order for the processor is used.
35
36     THE CIPHER INTERFACE
37
38     The cipher interface is implemented as an array of bytes in which
39     ↪lower
40     AES bit sequence indexes map to higher numeric significance within
41     ↪bytes.
42
43     uint_8t                (an unsigned 8-bit type)
44     uint_32t              (an unsigned 32-bit type)
45     struct aes_encrypt_ctx (structure for the cipher encryption
46     ↪context)
47     struct aes_decrypt_ctx (structure for the cipher decryption
48     ↪context)
49     AES_RETURN             the function return type
50
51     C subroutine calls:
52
53     AES_RETURN aes_encrypt_key128(const unsigned char *key,
54     ↪aes_encrypt_ctx cx[1]);
55     AES_RETURN aes_encrypt_key192(const unsigned char *key,
56     ↪aes_encrypt_ctx cx[1]);
57     AES_RETURN aes_encrypt_key256(const unsigned char *key,
58     ↪aes_encrypt_ctx cx[1]);
59     AES_RETURN aes_encrypt(const unsigned char *in, unsigned char *out,
60     ↪const
61     ↪aes_encrypt_ctx cx[1]);
62
63     AES_RETURN aes_decrypt_key128(const unsigned char *key,
64     ↪aes_decrypt_ctx cx[1]);
65     AES_RETURN aes_decrypt_key192(const unsigned char *key,
66     ↪aes_decrypt_ctx cx[1]);
67     AES_RETURN aes_decrypt_key256(const unsigned char *key,
68     ↪aes_decrypt_ctx cx[1]);
69     AES_RETURN aes_decrypt(const unsigned char *in, unsigned char *out,
70     ↪const
71     ↪aes_decrypt_ctx cx[1]);
72
73     IMPORTANT NOTE: If you are using this C interface with dynamic
74     ↪tables make sure that
75     you call aes_init() before AES is used so that the tables are
76     ↪initialised.
77
78     C++ aes class subroutines:
79
80     Class AESencrypt for encryption

```

```

67     Constructors:
68         AESencrypt(void)
69         AESencrypt(const unsigned char *key) - 128 bit key
70     Members:
71         AES_RETURN key128(const unsigned char *key)
72         AES_RETURN key192(const unsigned char *key)
73         AES_RETURN key256(const unsigned char *key)
74         AES_RETURN encrypt(const unsigned char *in, unsigned char *
→out) const
75
76     Class AESdecrypt for encryption
77     Constructors:
78         AESdecrypt(void)
79         AESdecrypt(const unsigned char *key) - 128 bit key
80     Members:
81         AES_RETURN key128(const unsigned char *key)
82         AES_RETURN key192(const unsigned char *key)
83         AES_RETURN key256(const unsigned char *key)
84         AES_RETURN decrypt(const unsigned char *in, unsigned char *
→out) const
85     */
86
87     #if !defined( _AESOPT_H )
88     #define _AESOPT_H
89
90     #if defined( __cplusplus )
91     #include "aescpp.h"
92     #else
93     #include "aes.h"
94     #endif
95
96     /* PLATFORM SPECIFIC INCLUDES */
97
98     #include "brg_endian.h"
99
100    /* CONFIGURATION - THE USE OF DEFINES
101
102        Later in this section there are a number of defines that control
→the
103        operation of the code. In each section, the purpose of each
→define is
104        explained so that the relevant form can be included or excluded
→by
105        setting either 1's or 0's respectively on the branches of the
→related
106        #if clauses. The following local defines should not be changed.
107    */
108
109    #define ENCRYPTION_IN_C 1

```

```

110 #define DECRYPTION_IN_C      2
111 #define ENC_KEYING_IN_C      4
112 #define DEC_KEYING_IN_C      8
113
114 #define NO_TABLES             0
115 #define ONE_TABLE             1
116 #define FOUR_TABLES          4
117 #define NONE                  0
118 #define PARTIAL               1
119 #define FULL                  2
120
121 /* --- START OF USER CONFIGURED OPTIONS --- */
122
123 /* 1. BYTE ORDER WITHIN 32 BIT WORDS
124
125     The fundamental data processing units in Rijndael are 8-bit bytes
126     ↪. The
127     input, output and key input are all enumerated arrays of bytes in
128     ↪ which
129     bytes are numbered starting at zero and increasing to one less
130     ↪ than the
131     number of bytes in the array in question. This enumeration is
132     ↪ only used
133     for naming bytes and does not imply any adjacency or order
134     ↪ relationship
135     from one byte to another. When these inputs and outputs are
136     ↪ considered
137     as bit sequences, bits  $8*n$  to  $8*n+7$  of the bit sequence are
138     ↪ mapped to
139     byte[n] with bit  $8n+i$  in the sequence mapped to bit  $7-i$  within
140     ↪ the byte.
141     In this implementation bits are numbered from 0 to 7 starting at
142     ↪ the
143     numerically least significant end of each byte (bit  $n$  represents
144     ↪  $2^n$ ).
145
146     However, Rijndael can be implemented more efficiently using 32-
147     ↪ bit
148     words by packing bytes into words so that bytes  $4*n$  to  $4*n+3$  are
149     ↪ placed
150     into word[n]. While in principle these bytes can be assembled
151     ↪ into words
152     in any positions, this implementation only supports the two
153     ↪ formats in
154     which bytes in adjacent positions within words also have adjacent
155     ↪ byte
156     numbers. This order is called big-endian if the lowest numbered
157     ↪ bytes
158     in words have the highest numeric significance and little-endian

```

```

143     ↪if the
144         opposite applies.
145
146     This code can work in either order irrespective of the order used
147     ↪by the
148     machine on which it runs. Normally the internal byte order will
149     ↪be set
150     to the order of the processor on which the code is to be run but
151     ↪this
152     define can be used to reverse this in special situations
153
154     WARNING: Assembler code versions rely on PLATFORM_BYTE_ORDER
155     ↪being set.
156     This define will hence be redefined later (in section 4) if
157     ↪necessary
158     */
159
160 #if 1
161 # define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
162 #elif 0
163 # define ALGORITHM_BYTE_ORDER IS_LITTLE_ENDIAN
164 #elif 0
165 # define ALGORITHM_BYTE_ORDER IS_BIG_ENDIAN
166 #else
167 # error The algorithm byte order is not defined
168 #endif
169
170 /* 2. VIA ACE SUPPORT */
171
172 #if defined( __GNUC__ ) && defined( __i386__ ) \
173     || defined( _WIN32 ) && defined( _M_IX86 ) \
174     && !(defined( _WIN64 ) || defined( _WIN32_WCE ) || defined( _MSC_VER
175     ↪ ) && ( _MSC_VER <= 800 ))
176 # define VIA_ACE_POSSIBLE
177 #endif
178
179 /* Define this option if support for the VIA ACE is required. This
180 ↪uses
181     inline assembler instructions and is only implemented for the
182     ↪Microsoft,
183     Intel and GCC compilers. If VIA ACE is known to be present, then
184     ↪defining
185     ASSUME_VIA_ACE_PRESENT will remove the ordinary encryption/
186     ↪decryption
187     code. If USE_VIA_ACE_IF_PRESENT is defined then VIA ACE will be
188     ↪used if
189     it is detected (both present and enabled) but the normal AES code
190     ↪will
191     also be present.

```

```

179
180     When VIA ACE is to be used, all AES encryption contexts MUST be
↪16 byte
181     aligned; other input/output buffers do not need to be 16 byte
↪aligned
182     but there are very large performance gains if this can be
↪arranged.
183     VIA ACE also requires the decryption key schedule to be in
↪reverse
184     order (which later checks below ensure).
185     */
186
187     #if 1 && defined( VIA_ACE_POSSIBLE ) && !defined(
↪USE_VIA_ACE_IF_PRESENT )
188     #   define USE_VIA_ACE_IF_PRESENT
189     #endif
190
191     #if 0 && defined( VIA_ACE_POSSIBLE ) && !defined(
↪ASSUME_VIA_ACE_PRESENT )
192     #   define ASSUME_VIA_ACE_PRESENT
193     #   endif
194
195     /* 3. ASSEMBLER SUPPORT
196
197     This define (which can be on the command line) enables the use of
↪ the
198     assembler code routines for encryption, decryption and key
↪scheduling
199     as follows:
200
201     ASM_X86_V1C uses the assembler (aes_x86_v1.asm) with large tables
↪ for
202                                encryption and decryption and but with key scheduling
↪ in C
203     ASM_X86_V2  uses assembler (aes_x86_v2.asm) with compressed
↪tables for
204                                encryption, decryption and key scheduling
205     ASM_X86_V2C uses assembler (aes_x86_v2.asm) with compressed
↪tables for
206                                encryption and decryption and but with key scheduling
↪ in C
207     ASM_AMD64_C uses assembler (aes_amd64.asm) with compressed tables
↪ for
208                                encryption and decryption and but with key scheduling
↪ in C
209
210     Change one 'if 0' below to 'if 1' to select the version or define
211     as a compilation option.
212     */

```



```

213
214 #if 0 && !defined( ASM_X86_V1C )
215 #   define ASM_X86_V1C
216 #elif 0 && !defined( ASM_X86_V2 )
217 #   define ASM_X86_V2
218 #elif 0 && !defined( ASM_X86_V2C )
219 #   define ASM_X86_V2C
220 #elif 0 && !defined( ASM_AMD64_C )
221 #   define ASM_AMD64_C
222 #endif
223
224 #if (defined( ASM_X86_V1C ) || defined( ASM_X86_V2 ) || defined(
↪ASM_X86_V2C )) \
225     && !defined( _M_IX86 ) || defined( ASM_AMD64_C ) && !defined(
↪_M_X64 )
226 #   error Assembler code is only available for x86 and AMD64 systems
227 #endif
228
229 /* 4. FAST INPUT/OUTPUT OPERATIONS.
230
231     On some machines it is possible to improve speed by transferring
↪the
232     bytes in the input and output arrays to and from the internal 32-
↪bit
233     variables by addressing these arrays as if they are arrays of 32-
↪bit
234     words. On some machines this will always be possible but there
↪may
235     be a large performance penalty if the byte arrays are not aligned
↪on
236     the normal word boundaries. On other machines this technique will
237     lead to memory access errors when such 32-bit word accesses are
↪not
238     properly aligned. The option SAFE_IO avoids such problems but
↪will
239     often be slower on those machines that support misaligned access
240     (especially so if care is taken to align the input and output
↪byte
241     arrays on 32-bit word boundaries). If SAFE_IO is not defined it
↪is
242     assumed that access to byte arrays as if they are arrays of 32-
↪bit
243     words will not cause problems when such accesses are misaligned.
244 */
245 #if 1 && !defined( _MSC_VER )
246 #   define SAFE_IO
247 #endif
248
249 /* 5. LOOP UNROLLING

```

```

250
251     The code for encryption and decryption cycles through a number of
    ↪ rounds
252     that can be implemented either in a loop or by expanding the code
    ↪ into a
253     long sequence of instructions, the latter producing a larger
    ↪ program but
254     one that will often be much faster. The latter is called loop
    ↪ unrolling.
255     There are also potential speed advantages in expanding two
    ↪ iterations in
256     a loop with half the number of iterations, which is called
    ↪ partial loop
257     unrolling. The following options allow partial or full loop
    ↪ unrolling
258     to be set independently for encryption and decryption
259 */
260 #if 1
261 #   define ENC_UNROLL    FULL
262 #elif 0
263 #   define ENC_UNROLL    PARTIAL
264 #else
265 #   define ENC_UNROLL    NONE
266 #endif
267
268 #if 1
269 #   define DEC_UNROLL    FULL
270 #elif 0
271 #   define DEC_UNROLL    PARTIAL
272 #else
273 #   define DEC_UNROLL    NONE
274 #endif
275
276 #if 1
277 #   define ENC_KS_UNROLL
278 #endif
279
280 #if 1
281 #   define DEC_KS_UNROLL
282 #endif
283
284 /* 6. FAST FINITE FIELD OPERATIONS
285
286     If this section is included, tables are used to provide faster
    ↪ finite
287     field arithmetic (this has no effect if FIXED_TABLES is defined).
288 */
289 #if 1
290 #   define FF_TABLES

```

```

291 #endif
292
293 /* 7. INTERNAL STATE VARIABLE FORMAT
294
295     The internal state of Rijndael is stored in a number of local 32-
↪bit
296     word variables which can be defined either as an array or as
↪individual
297     names variables. Include this section if you want to store these
↪local
298     variables in arrays. Otherwise individual local variables will be
↪used.
299 */
300 #if 1
301 # define ARRAYS
302 #endif
303
304 /* 8. FIXED OR DYNAMIC TABLES
305
306     When this section is included the tables used by the code are
↪compiled
307     statically into the binary file. Otherwise the subroutine
↪aes_init()
308     must be called to compute them before the code is first used.
309 */
310 #if 1 && !(defined( _MSC_VER ) && ( _MSC_VER <= 800 ))
311 # define FIXED_TABLES
312 #endif
313
314 /* 9. MASKING OR CASTING FROM LONGER VALUES TO BYTES
315
316     In some systems it is better to mask longer values to extract
↪bytes
317     rather than using a cast. This option allows this choice.
318 */
319 #if 0
320 # define to_byte(x) ((uint_8t)(x))
321 #else
322 # define to_byte(x) ((x) & 0xff)
323 #endif
324
325 /* 10. TABLE ALIGNMENT
326
327     On some systems speed will be improved by aligning the AES large
↪lookup
328     tables on particular boundaries. This define should be set to a
↪power of
329     two giving the desired alignment. It can be left undefined if
↪alignment

```

```

330      is not needed. This option is specific to the Microsoft VC++
    ↪ compiler -
331      it seems to sometimes cause trouble for the VC++ version 6
    ↪ compiler.
332  */
333
334  #if 1 && defined( _MSC_VER ) && ( _MSC_VER >= 1300 )
335  #   define TABLE_ALIGN 32
336  #endif
337
338  /* 11. REDUCE CODE AND TABLE SIZE
339
340      This replaces some expanded macros with function calls if
    ↪ AES_ASM_V2 or
341      AES_ASM_V2C are defined
342  */
343
344  #if 1 && (defined( ASM_X86_V2 ) || defined( ASM_X86_V2C ))
345  #   define REDUCE_CODE_SIZE
346  #endif
347
348  /* 12. TABLE OPTIONS
349
350      This cipher proceeds by repeating in a number of cycles known as
    ↪ 'rounds'
351      which are implemented by a round function which can optionally be
    ↪ speeded
352      up using tables. The basic tables are each 256 32-bit words,
    ↪ with either
353      one or four tables being required for each round function
    ↪ depending on
354      how much speed is required. The encryption and decryption round
    ↪ functions
355      are different and the last encryption and decryption round
    ↪ functions are
356      different again making four different round functions in all.
357
358      This means that:
359      1. Normal encryption and decryption rounds can each use either
    ↪ 0, 1
360          or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
361      2. The last encryption and decryption rounds can also use
    ↪ either 0, 1
362          or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
363
364      Include or exclude the appropriate definitions below to set the
    ↪ number
365      of tables used by this implementation.
366  */

```

```

367
368 /* set tables for the normal encryption round */
369 # define ENC_ROUND    FOUR_TABLES
370 #elif 0
371 # define ENC_ROUND    ONE_TABLE
372 #else
373 # define ENC_ROUND    NO_TABLES
374 #endif
375
376 /* set tables for the last encryption round */
377 # define LAST_ENC_ROUND    FOUR_TABLES
378 #elif 0
379 # define LAST_ENC_ROUND    ONE_TABLE
380 #else
381 # define LAST_ENC_ROUND    NO_TABLES
382 #endif
383
384 /* set tables for the normal decryption round */
385 # define DEC_ROUND    FOUR_TABLES
386 #elif 0
387 # define DEC_ROUND    ONE_TABLE
388 #else
389 # define DEC_ROUND    NO_TABLES
390 #endif
391
392 /* set tables for the last decryption round */
393 # define LAST_DEC_ROUND    FOUR_TABLES
394 #elif 0
395 # define LAST_DEC_ROUND    ONE_TABLE
396 #else
397 # define LAST_DEC_ROUND    NO_TABLES
398 #endif
399
400 /* The decryption key schedule can be speeded up with tables in the
401 ↪same
402 way that the round functions can. Include or exclude the
403 ↪following
404 defines to set this requirement.
405 */
406 #if 1
407 # define KEY_SCHED    FOUR_TABLES
408 #elif 0
409 # define KEY_SCHED    ONE_TABLE
410 #else
411 # define KEY_SCHED    NO_TABLES
412 #endif
413
414 /* ---- END OF USER CONFIGURED OPTIONS ---- */

```

```

414  /* VIA ACE support is only available for VC++ and GCC */
415
416  #if !defined( _MSC_VER ) && !defined( __GNUC__ )
417  #  if defined( ASSUME_VIA_ACE_PRESENT )
418  #    undef ASSUME_VIA_ACE_PRESENT
419  #  endif
420  #  if defined( USE_VIA_ACE_IF_PRESENT )
421  #    undef USE_VIA_ACE_IF_PRESENT
422  #  endif
423  #endif
424
425  #if defined( ASSUME_VIA_ACE_PRESENT ) && !defined(
    ↪USE_VIA_ACE_IF_PRESENT )
426  #  define USE_VIA_ACE_IF_PRESENT
427  #endif
428
429  #if defined( USE_VIA_ACE_IF_PRESENT ) && !defined ( AES_REV_DKS )
430  #  define AES_REV_DKS
431  #endif
432
433  /* Assembler support requires the use of platform byte order */
434
435  #if ( defined( ASM_X86_V1C ) || defined( ASM_X86_V2C ) || defined(
    ↪ASM_AMD64_C ) ) \
436      && (ALGORITHM_BYTE_ORDER != PLATFORM_BYTE_ORDER)
437  #  undef ALGORITHM_BYTE_ORDER
438  #  define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
439  #endif
440
441  /* In this implementation the columns of the state array are each
    ↪held in
442      32-bit words. The state array can be held in various ways: in an
    ↪array
443      of words, in a number of individual word variables or in a number
    ↪of
444      processor registers. The following define maps a variable name x
    ↪and
445      a column number c to the way the state array variable is to be
    ↪held.
446      The first define below maps the state into an array x[c] whereas
    ↪the
447      second form maps the state into a number of individual variables
    ↪x0,
448      x1, etc. Another form could map individual state columns to
    ↪machine
449      register names.
450  */
451
452  #if defined( ARRAYS )

```

```

453 # define s(x,c) x[c]
454 #else
455 # define s(x,c) x##c
456 #endif
457
458 /* This implementation provides subroutines for encryption,
459 → decryption
460 and for setting the three key lengths (separately) for encryption
461 and decryption. Since not all functions are needed, masks are set
462 up here to determine which will be implemented in C
463 */
464
465 #if !defined( AES_ENCRYPT )
466 # define EFUNCS_IN_C 0
467 #elif defined( ASSUME_VIA_ACE_PRESENT ) || defined( ASM_X86_V1C ) \
468 || defined( ASM_X86_V2C ) || defined( ASM_AMD64_C )
469 # define EFUNCS_IN_C ENC_KEYING_IN_C
470 #elif !defined( ASM_X86_V2 )
471 # define EFUNCS_IN_C ( ENCRYPTION_IN_C | ENC_KEYING_IN_C )
472 #else
473 # define EFUNCS_IN_C 0
474 #endif
475
476 #if !defined( AES_DECRYPT )
477 # define DFUNCS_IN_C 0
478 #elif defined( ASSUME_VIA_ACE_PRESENT ) || defined( ASM_X86_V1C ) \
479 || defined( ASM_X86_V2C ) || defined( ASM_AMD64_C )
480 # define DFUNCS_IN_C DEC_KEYING_IN_C
481 #elif !defined( ASM_X86_V2 )
482 # define DFUNCS_IN_C ( DECRYPTION_IN_C | DEC_KEYING_IN_C )
483 #else
484 # define DFUNCS_IN_C 0
485 #endif
486
487 #define FUNCS_IN_C ( EFUNCS_IN_C | DFUNCS_IN_C )
488
489 /* END OF CONFIGURATION OPTIONS */
490
491 #define RC_LENGTH (5 * (AES_BLOCK_SIZE / 4 - 2))
492
493 /* Disable or report errors on some combinations of options */
494
495 #if ENC_ROUND == NO_TABLES && LAST_ENC_ROUND != NO_TABLES
496 # undef LAST_ENC_ROUND
497 # define LAST_ENC_ROUND NO_TABLES
498 #elif ENC_ROUND == ONE_TABLE && LAST_ENC_ROUND == FOUR_TABLES
499 # undef LAST_ENC_ROUND
500 # define LAST_ENC_ROUND ONE_TABLE
501 #endif

```

```

501
502 #if ENC_ROUND == NO_TABLES && ENC_UNROLL != NONE
503 #   undef ENC_UNROLL
504 #   define ENC_UNROLL NONE
505 #endif
506
507 #if DEC_ROUND == NO_TABLES && LAST_DEC_ROUND != NO_TABLES
508 #   undef LAST_DEC_ROUND
509 #   define LAST_DEC_ROUND NO_TABLES
510 #elif DEC_ROUND == ONE_TABLE && LAST_DEC_ROUND == FOUR_TABLES
511 #   undef LAST_DEC_ROUND
512 #   define LAST_DEC_ROUND ONE_TABLE
513 #endif
514
515 #if DEC_ROUND == NO_TABLES && DEC_UNROLL != NONE
516 #   undef DEC_UNROLL
517 #   define DEC_UNROLL NONE
518 #endif
519
520 #if defined( bswap32 )
521 #   define aes_sw32      bswap32
522 #elif defined( bswap_32 )
523 #   define aes_sw32      bswap_32
524 #else
525 #   define brot(x,n)      (((uint_32t)(x) <<  n) | ((uint_32t)(x) >> (32
526 ↪- n)))
527 #   define aes_sw32(x) ((brot((x),8) & 0x00ff00ff) | (brot((x),24) & 0
528 ↪x00ff00ff))
529 #endif
530
531 /* upr(x,n):  rotates bytes within words by n positions, moving
532 ↪bytes to
533                higher index positions with wrap around into low
534 ↪positions
535                ups(x,n):  moves bytes by n positions to higher index positions
536 ↪in
537                words but without wrap around
538                bval(x,n):  extracts a byte from a word
539
540                WARNING:  The definitions given here are intended only for use
541 ↪with
542                unsigned variables and with shift counts that are
543 ↪compile
544                time constants
545 */
546
547 #if ( ALGORITHM_BYTE_ORDER == IS_LITTLE_ENDIAN )
548 #   define upr(x,n)      (((uint_32t)(x) << (8 * (n))) | ((uint_32t)(x
549 ↪) >> (32 - 8 * (n))))

```



```

542 # define ups(x,n)      ((uint_32t) (x) << (8 * (n)))
543 # define bval(x,n)      to_byte((x) >> (8 * (n)))
544 # define bytes2word(b0, b1, b2, b3) \
545     (((uint_32t)(b3) << 24) | ((uint_32t)(b2) << 16) | ((uint_32t
↪)(b1) << 8) | (b0))
546 #endif
547
548 #if ( ALGORITHM_BYTE_ORDER == IS_BIG_ENDIAN )
549 # define upr(x,n)      (((uint_32t)(x) >> (8 * (n))) | ((uint_32t)(x
↪) << (32 - 8 * (n))))
550 # define ups(x,n)      ((uint_32t) (x) >> (8 * (n)))
551 # define bval(x,n)      to_byte((x) >> (24 - 8 * (n)))
552 # define bytes2word(b0, b1, b2, b3) \
553     (((uint_32t)(b0) << 24) | ((uint_32t)(b1) << 16) | ((uint_32t
↪)(b2) << 8) | (b3))
554 #endif
555
556 #if defined( SAFE_IO )
557 # define word_in(x,c)    bytes2word(((const uint_8t*)(x)+4*c)[0], ((
↪const uint_8t*)(x)+4*c)[1], \
558                                     ((const uint_8t*)(x)+4*c)[2], ((
↪const uint_8t*)(x)+4*c)[3])
559 # define word_out(x,c,v) { ((uint_8t*)(x)+4*c)[0] = bval(v,0); ((
↪uint_8t*)(x)+4*c)[1] = bval(v,1); \
560                             ((uint_8t*)(x)+4*c)[2] = bval(v,2); ((
↪uint_8t*)(x)+4*c)[3] = bval(v,3); }
561 #elif ( ALGORITHM_BYTE_ORDER == PLATFORM_BYTE_ORDER )
562 # define word_in(x,c)    (*((uint_32t*)(x)+(c)))
563 # define word_out(x,c,v) (*((uint_32t*)(x)+(c)) = (v))
564 #else
565 # define word_in(x,c)    aes_sw32 (*((uint_32t*)(x)+(c)))
566 # define word_out(x,c,v) (*((uint_32t*)(x)+(c)) = aes_sw32(v))
567 #endif
568
569 /* the finite field modular polynomial and elements */
570
571 #define WPOLY    0x011b
572 #define BPOLY    0x1b
573
574 /* multiply four bytes in GF(2^8) by 'x' {02} in parallel */
575
576 #define gf_c1    0x80808080
577 #define gf_c2    0x7f7f7f7f
578 #define gf_mulx(x)  (((x) & gf_c2) << 1) ^ (((x) & gf_c1) >> 7) *
↪BPOLY))
579
580 /* The following defines provide alternative definitions of gf_mulx
↪that might
581     give improved performance if a fast 32-bit multiply is not

```

```

→available. Note
582     that a temporary variable u needs to be defined where gf_mulx is
→used.
583
584 #define gf_mulx(x) (u = (x) & gf_c1, u != (u >> 1), ((x) & gf_c2) <<
→1) ^ ((u >> 3) | (u >> 6))
585 #define gf_c4 (0x01010101 * BPOLY)
586 #define gf_mulx(x) (u = (x) & gf_c1, ((x) & gf_c2) << 1) ^ ((u - (u
→>> 7)) & gf_c4)
587 */
588
589 /* Work out which tables are needed for the different options */
590
591 #if defined( ASM_X86_V1C )
592 #   if defined( ENC_ROUND )
593 #       undef ENC_ROUND
594 #   endif
595 #   define ENC_ROUND    FOUR_TABLES
596 #   if defined( LAST_ENC_ROUND )
597 #       undef LAST_ENC_ROUND
598 #   endif
599 #   define LAST_ENC_ROUND    FOUR_TABLES
600 #   if defined( DEC_ROUND )
601 #       undef DEC_ROUND
602 #   endif
603 #   define DEC_ROUND    FOUR_TABLES
604 #   if defined( LAST_DEC_ROUND )
605 #       undef LAST_DEC_ROUND
606 #   endif
607 #   define LAST_DEC_ROUND    FOUR_TABLES
608 #   if defined( KEY_SCHED )
609 #       undef KEY_SCHED
610 #       define KEY_SCHED    FOUR_TABLES
611 #   endif
612 #endif
613
614 #if ( FUNCS_IN_C & ENCRYPTION_IN_C ) || defined( ASM_X86_V1C )
615 #   if ENC_ROUND == ONE_TABLE
616 #       define FT1_SET
617 #   elif ENC_ROUND == FOUR_TABLES
618 #       define FT4_SET
619 #   else
620 #       define SBX_SET
621 #   endif
622 #   if LAST_ENC_ROUND == ONE_TABLE
623 #       define FL1_SET
624 #   elif LAST_ENC_ROUND == FOUR_TABLES
625 #       define FL4_SET
626 #   elif !defined( SBX_SET )

```

```

627 #     define SBX_SET
628 # endif
629 #endif
630
631 #if ( FUNCS_IN_C & DECRYPTION_IN_C ) || defined( ASM_X86_V1C )
632 # if DEC_ROUND == ONE_TABLE
633 #     define IT1_SET
634 # elif DEC_ROUND == FOUR_TABLES
635 #     define IT4_SET
636 # else
637 #     define ISB_SET
638 # endif
639 # if LAST_DEC_ROUND == ONE_TABLE
640 #     define IL1_SET
641 # elif LAST_DEC_ROUND == FOUR_TABLES
642 #     define IL4_SET
643 # elif !defined( ISB_SET )
644 #     define ISB_SET
645 # endif
646 #endif
647
648 #if !(defined( REDUCE_CODE_SIZE ) && (defined( ASM_X86_V2 ) ||
↳defined( ASM_X86_V2C )))
649 # if ((FUNCS_IN_C & ENC_KEYING_IN_C) || (FUNCS_IN_C &
↳DEC_KEYING_IN_C))
650 #     if KEY_SCHED == ONE_TABLE
651 #         if !defined( FL1_SET ) && !defined( FL4_SET )
652 #             define LS1_SET
653 #         endif
654 #     elif KEY_SCHED == FOUR_TABLES
655 #         if !defined( FL4_SET )
656 #             define LS4_SET
657 #         endif
658 #     elif !defined( SBX_SET )
659 #         define SBX_SET
660 #     endif
661 # endif
662 # if (FUNCS_IN_C & DEC_KEYING_IN_C)
663 #     if KEY_SCHED == ONE_TABLE
664 #         define IM1_SET
665 #     elif KEY_SCHED == FOUR_TABLES
666 #         define IM4_SET
667 #     elif !defined( SBX_SET )
668 #         define SBX_SET
669 #     endif
670 # endif
671 #endif
672
673 /* generic definitions of Rijndael macros that use tables */

```

```

674
675 #define no_table(x,box,vf,rf,c) bytes2word( \
676     box[bval(vf(x,0,c),rf(0,c))], \
677     box[bval(vf(x,1,c),rf(1,c))], \
678     box[bval(vf(x,2,c),rf(2,c))], \
679     box[bval(vf(x,3,c),rf(3,c))])
680
681 #define one_table(x,op,tab,vf,rf,c) \
682 (    tab[bval(vf(x,0,c),rf(0,c))] \
683 ^ op(tab[bval(vf(x,1,c),rf(1,c))],1) \
684 ^ op(tab[bval(vf(x,2,c),rf(2,c))],2) \
685 ^ op(tab[bval(vf(x,3,c),rf(3,c))],3))
686
687 #define four_tables(x,tab,vf,rf,c) \
688 (    tab[0][bval(vf(x,0,c),rf(0,c))] \
689 ^ tab[1][bval(vf(x,1,c),rf(1,c))] \
690 ^ tab[2][bval(vf(x,2,c),rf(2,c))] \
691 ^ tab[3][bval(vf(x,3,c),rf(3,c))])
692
693 #define vf1(x,r,c)    (x)
694 #define rf1(r,c)      (r)
695 #define rf2(r,c)      ((8+r-c)&3)
696
697 /* perform forward and inverse column mix operation on four bytes in
698 ↪long word x in */
699 /* parallel. NOTE: x must be a simple variable, NOT an expression in
700 ↪these macros. */
701
702 #if !(defined( REDUCE_CODE_SIZE ) && (defined( ASM_X86_V2 ) ||
703     ↪defined( ASM_X86_V2C )))
704
705 #if defined( FM4_SET )           /* not currently used */
706 #   define fwd_mcol(x)          four_tables(x,t_use(f,m),vf1,rf1,0)
707 #elif defined( FM1_SET )        /* not currently used */
708 #   define fwd_mcol(x)          one_table(x,upr,t_use(f,m),vf1,rf1,0)
709 #else
710 #   define dec_fmvars            uint_32t g2
711 #   define fwd_mcol(x)          (g2 = gf_mulx(x), g2 ^ upr((x) ^ g2, 3) ^
712     ↪upr((x), 2) ^ upr((x), 1))
713 #endif
714
715 #if defined( IM4_SET )
716 #   define inv_mcol(x)          four_tables(x,t_use(i,m),vf1,rf1,0)
717 #elif defined( IM1_SET )
718 #   define inv_mcol(x)          one_table(x,upr,t_use(i,m),vf1,rf1,0)
719 #else
720 #   define dec_imvars            uint_32t g2, g4, g9
721 #   define inv_mcol(x)          (g2 = gf_mulx(x), g4 = gf_mulx(g2), g9 =
722     ↪(x) ^ gf_mulx(g4), g4 ^= g9, \

```

```

718             (x) ^ g2 ^ g4 ^ upr(g2 ^ g9, 3) ^ upr(g4,
719             ↪ 2) ^ upr(g9, 1))
719 #endif
720
721 #if defined( FL4_SET )
722 #   define ls_box(x,c)         four_tables(x,t_use(f,l),vf1,rf2,c)
723 #elif defined( LS4_SET )
724 #   define ls_box(x,c)         four_tables(x,t_use(l,s),vf1,rf2,c)
725 #elif defined( FL1_SET )
726 #   define ls_box(x,c)         one_table(x,upr,t_use(f,l),vf1,rf2,c)
727 #elif defined( LS1_SET )
728 #   define ls_box(x,c)         one_table(x,upr,t_use(l,s),vf1,rf2,c)
729 #else
730 #   define ls_box(x,c)         no_table(x,t_use(s,box),vf1,rf2,c)
731 #endif
732
733 #endif
734
735 #if defined( ASM_X86_V1C ) && defined( AES_DECRYPT ) && !defined(
736 ↪ ISB_SET )
736 #   define ISB_SET
737 #endif
738
739 #endif

```

src/Obf/aestab.c

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007

```

```

19  */
20
21  #define DO_TABLES
22
23  #include "aes.h"
24  #include "aesopt.h"
25
26  #if defined(FIXED_TABLES)
27
28  #define sb_data(w) {\
29      w(0x63), w(0x7c), w(0x77), w(0x7b), w(0xf2), w(0x6b), w(0x6f), w
    ↪(0xc5),\
30      w(0x30), w(0x01), w(0x67), w(0x2b), w(0xfe), w(0xd7), w(0xab), w
    ↪(0x76),\
31      w(0xca), w(0x82), w(0xc9), w(0x7d), w(0xfa), w(0x59), w(0x47), w
    ↪(0xf0),\
32      w(0xad), w(0xd4), w(0xa2), w(0xaf), w(0x9c), w(0xa4), w(0x72), w
    ↪(0xc0),\
33      w(0xb7), w(0xfd), w(0x93), w(0x26), w(0x36), w(0x3f), w(0xf7), w
    ↪(0xcc),\
34      w(0x34), w(0xa5), w(0xe5), w(0xf1), w(0x71), w(0xd8), w(0x31), w
    ↪(0x15),\
35      w(0x04), w(0xc7), w(0x23), w(0xc3), w(0x18), w(0x96), w(0x05), w
    ↪(0x9a),\
36      w(0x07), w(0x12), w(0x80), w(0xe2), w(0xeb), w(0x27), w(0xb2), w
    ↪(0x75),\
37      w(0x09), w(0x83), w(0x2c), w(0x1a), w(0x1b), w(0x6e), w(0x5a), w
    ↪(0xa0),\
38      w(0x52), w(0x3b), w(0xd6), w(0xb3), w(0x29), w(0xe3), w(0x2f), w
    ↪(0x84),\
39      w(0x53), w(0xd1), w(0x00), w(0xed), w(0x20), w(0xfc), w(0xb1), w
    ↪(0x5b),\
40      w(0x6a), w(0xcb), w(0xbe), w(0x39), w(0x4a), w(0x4c), w(0x58), w
    ↪(0xcf),\
41      w(0xd0), w(0xef), w(0xaa), w(0xfb), w(0x43), w(0x4d), w(0x33), w
    ↪(0x85),\
42      w(0x45), w(0xf9), w(0x02), w(0x7f), w(0x50), w(0x3c), w(0x9f), w
    ↪(0xa8),\
43      w(0x51), w(0xa3), w(0x40), w(0x8f), w(0x92), w(0x9d), w(0x38), w
    ↪(0xf5),\
44      w(0xbc), w(0xb6), w(0xda), w(0x21), w(0x10), w(0xff), w(0xf3), w
    ↪(0xd2),\
45      w(0xcd), w(0x0c), w(0x13), w(0xec), w(0x5f), w(0x97), w(0x44), w
    ↪(0x17),\
46      w(0xc4), w(0xa7), w(0x7e), w(0x3d), w(0x64), w(0x5d), w(0x19), w
    ↪(0x73),\
47      w(0x60), w(0x81), w(0x4f), w(0xdc), w(0x22), w(0x2a), w(0x90), w
    ↪(0x88),\
48      w(0x46), w(0xee), w(0xb8), w(0x14), w(0xde), w(0x5e), w(0x0b), w

```

```

    ↪(0xdb),\
49     w(0xe0), w(0x32), w(0x3a), w(0x0a), w(0x49), w(0x06), w(0x24), w
    ↪(0x5c),\
50     w(0xc2), w(0xd3), w(0xac), w(0x62), w(0x91), w(0x95), w(0xe4), w
    ↪(0x79),\
51     w(0xe7), w(0xc8), w(0x37), w(0x6d), w(0x8d), w(0xd5), w(0x4e), w
    ↪(0xa9),\
52     w(0x6c), w(0x56), w(0xf4), w(0xea), w(0x65), w(0x7a), w(0xae), w
    ↪(0x08),\
53     w(0xba), w(0x78), w(0x25), w(0x2e), w(0x1c), w(0xa6), w(0xb4), w
    ↪(0xc6),\
54     w(0xe8), w(0xdd), w(0x74), w(0x1f), w(0x4b), w(0xbd), w(0x8b), w
    ↪(0x8a),\
55     w(0x70), w(0x3e), w(0xb5), w(0x66), w(0x48), w(0x03), w(0xf6), w
    ↪(0x0e),\
56     w(0x61), w(0x35), w(0x57), w(0xb9), w(0x86), w(0xc1), w(0x1d), w
    ↪(0x9e),\
57     w(0xe1), w(0xf8), w(0x98), w(0x11), w(0x69), w(0xd9), w(0x8e), w
    ↪(0x94),\
58     w(0x9b), w(0x1e), w(0x87), w(0xe9), w(0xce), w(0x55), w(0x28), w
    ↪(0xdf),\
59     w(0x8c), w(0xa1), w(0x89), w(0x0d), w(0xbf), w(0xe6), w(0x42), w
    ↪(0x68),\
60     w(0x41), w(0x99), w(0x2d), w(0x0f), w(0xb0), w(0x54), w(0xbb), w
    ↪(0x16) }
61
62 #define isb_data(w) {\
63     w(0x52), w(0x09), w(0x6a), w(0xd5), w(0x30), w(0x36), w(0xa5), w
    ↪(0x38),\
64     w(0xbf), w(0x40), w(0xa3), w(0x9e), w(0x81), w(0xf3), w(0xd7), w
    ↪(0xfb),\
65     w(0x7c), w(0xe3), w(0x39), w(0x82), w(0x9b), w(0x2f), w(0xff), w
    ↪(0x87),\
66     w(0x34), w(0x8e), w(0x43), w(0x44), w(0xc4), w(0xde), w(0xe9), w
    ↪(0xcb),\
67     w(0x54), w(0x7b), w(0x94), w(0x32), w(0xa6), w(0xc2), w(0x23), w
    ↪(0x3d),\
68     w(0xee), w(0x4c), w(0x95), w(0x0b), w(0x42), w(0xfa), w(0xc3), w
    ↪(0x4e),\
69     w(0x08), w(0x2e), w(0xa1), w(0x66), w(0x28), w(0xd9), w(0x24), w
    ↪(0xb2),\
70     w(0x76), w(0x5b), w(0xa2), w(0x49), w(0x6d), w(0x8b), w(0xd1), w
    ↪(0x25),\
71     w(0x72), w(0xf8), w(0xf6), w(0x64), w(0x86), w(0x68), w(0x98), w
    ↪(0x16),\
72     w(0xd4), w(0xa4), w(0x5c), w(0xcc), w(0x5d), w(0x65), w(0xb6), w
    ↪(0x92),\
73     w(0x6c), w(0x70), w(0x48), w(0x50), w(0xfd), w(0xed), w(0xb9), w
    ↪(0xda),\

```

```

74     w(0x5e), w(0x15), w(0x46), w(0x57), w(0xa7), w(0x8d), w(0x9d), w
    ↪(0x84),\
75     w(0x90), w(0xd8), w(0xab), w(0x00), w(0x8c), w(0xbc), w(0xd3), w
    ↪(0x0a),\
76     w(0xf7), w(0xe4), w(0x58), w(0x05), w(0xb8), w(0xb3), w(0x45), w
    ↪(0x06),\
77     w(0xd0), w(0x2c), w(0x1e), w(0x8f), w(0xca), w(0x3f), w(0x0f), w
    ↪(0x02),\
78     w(0xc1), w(0xaf), w(0xbd), w(0x03), w(0x01), w(0x13), w(0x8a), w
    ↪(0x6b),\
79     w(0x3a), w(0x91), w(0x11), w(0x41), w(0x4f), w(0x67), w(0xdc), w
    ↪(0xea),\
80     w(0x97), w(0xf2), w(0xcf), w(0xce), w(0xf0), w(0xb4), w(0xe6), w
    ↪(0x73),\
81     w(0x96), w(0xac), w(0x74), w(0x22), w(0xe7), w(0xad), w(0x35), w
    ↪(0x85),\
82     w(0xe2), w(0xf9), w(0x37), w(0xe8), w(0x1c), w(0x75), w(0xdf), w
    ↪(0x6e),\
83     w(0x47), w(0xf1), w(0x1a), w(0x71), w(0x1d), w(0x29), w(0xc5), w
    ↪(0x89),\
84     w(0x6f), w(0xb7), w(0x62), w(0x0e), w(0xaa), w(0x18), w(0xbe), w
    ↪(0x1b),\
85     w(0xfc), w(0x56), w(0x3e), w(0x4b), w(0xc6), w(0xd2), w(0x79), w
    ↪(0x20),\
86     w(0x9a), w(0xdb), w(0xc0), w(0xfe), w(0x78), w(0xcd), w(0x5a), w
    ↪(0xf4),\
87     w(0x1f), w(0xdd), w(0xa8), w(0x33), w(0x88), w(0x07), w(0xc7), w
    ↪(0x31),\
88     w(0xb1), w(0x12), w(0x10), w(0x59), w(0x27), w(0x80), w(0xec), w
    ↪(0x5f),\
89     w(0x60), w(0x51), w(0x7f), w(0xa9), w(0x19), w(0xb5), w(0x4a), w
    ↪(0x0d),\
90     w(0x2d), w(0xe5), w(0x7a), w(0x9f), w(0x93), w(0xc9), w(0x9c), w
    ↪(0xef),\
91     w(0xa0), w(0xe0), w(0x3b), w(0x4d), w(0xae), w(0x2a), w(0xf5), w
    ↪(0xb0),\
92     w(0xc8), w(0xeb), w(0xbb), w(0x3c), w(0x83), w(0x53), w(0x99), w
    ↪(0x61),\
93     w(0x17), w(0x2b), w(0x04), w(0x7e), w(0xba), w(0x77), w(0xd6), w
    ↪(0x26),\
94     w(0xe1), w(0x69), w(0x14), w(0x63), w(0x55), w(0x21), w(0x0c), w
    ↪(0x7d) }
95
96 #define mm_data(w) {\
97     w(0x00), w(0x01), w(0x02), w(0x03), w(0x04), w(0x05), w(0x06), w
    ↪(0x07),\
98     w(0x08), w(0x09), w(0x0a), w(0x0b), w(0x0c), w(0x0d), w(0x0e), w
    ↪(0x0f),\
99     w(0x10), w(0x11), w(0x12), w(0x13), w(0x14), w(0x15), w(0x16), w

```



```

    ↪(0x17),\
100     w(0x18), w(0x19), w(0x1a), w(0x1b), w(0x1c), w(0x1d), w(0x1e), w
    ↪(0x1f),\
101     w(0x20), w(0x21), w(0x22), w(0x23), w(0x24), w(0x25), w(0x26), w
    ↪(0x27),\
102     w(0x28), w(0x29), w(0x2a), w(0x2b), w(0x2c), w(0x2d), w(0x2e), w
    ↪(0x2f),\
103     w(0x30), w(0x31), w(0x32), w(0x33), w(0x34), w(0x35), w(0x36), w
    ↪(0x37),\
104     w(0x38), w(0x39), w(0x3a), w(0x3b), w(0x3c), w(0x3d), w(0x3e), w
    ↪(0x3f),\
105     w(0x40), w(0x41), w(0x42), w(0x43), w(0x44), w(0x45), w(0x46), w
    ↪(0x47),\
106     w(0x48), w(0x49), w(0x4a), w(0x4b), w(0x4c), w(0x4d), w(0x4e), w
    ↪(0x4f),\
107     w(0x50), w(0x51), w(0x52), w(0x53), w(0x54), w(0x55), w(0x56), w
    ↪(0x57),\
108     w(0x58), w(0x59), w(0x5a), w(0x5b), w(0x5c), w(0x5d), w(0x5e), w
    ↪(0x5f),\
109     w(0x60), w(0x61), w(0x62), w(0x63), w(0x64), w(0x65), w(0x66), w
    ↪(0x67),\
110     w(0x68), w(0x69), w(0x6a), w(0x6b), w(0x6c), w(0x6d), w(0x6e), w
    ↪(0x6f),\
111     w(0x70), w(0x71), w(0x72), w(0x73), w(0x74), w(0x75), w(0x76), w
    ↪(0x77),\
112     w(0x78), w(0x79), w(0x7a), w(0x7b), w(0x7c), w(0x7d), w(0x7e), w
    ↪(0x7f),\
113     w(0x80), w(0x81), w(0x82), w(0x83), w(0x84), w(0x85), w(0x86), w
    ↪(0x87),\
114     w(0x88), w(0x89), w(0x8a), w(0x8b), w(0x8c), w(0x8d), w(0x8e), w
    ↪(0x8f),\
115     w(0x90), w(0x91), w(0x92), w(0x93), w(0x94), w(0x95), w(0x96), w
    ↪(0x97),\
116     w(0x98), w(0x99), w(0x9a), w(0x9b), w(0x9c), w(0x9d), w(0x9e), w
    ↪(0x9f),\
117     w(0xa0), w(0xa1), w(0xa2), w(0xa3), w(0xa4), w(0xa5), w(0xa6), w
    ↪(0xa7),\
118     w(0xa8), w(0xa9), w(0xaa), w(0xab), w(0xac), w(0xad), w(0xae), w
    ↪(0xaf),\
119     w(0xb0), w(0xb1), w(0xb2), w(0xb3), w(0xb4), w(0xb5), w(0xb6), w
    ↪(0xb7),\
120     w(0xb8), w(0xb9), w(0xba), w(0xbb), w(0xbc), w(0xbd), w(0xbe), w
    ↪(0xbf),\
121     w(0xc0), w(0xc1), w(0xc2), w(0xc3), w(0xc4), w(0xc5), w(0xc6), w
    ↪(0xc7),\
122     w(0xc8), w(0xc9), w(0xca), w(0xcb), w(0xcc), w(0xcd), w(0xce), w
    ↪(0xcf),\
123     w(0xd0), w(0xd1), w(0xd2), w(0xd3), w(0xd4), w(0xd5), w(0xd6), w
    ↪(0xd7),\

```

```

124     w(0xd8), w(0xd9), w(0xda), w(0xdb), w(0xdc), w(0xdd), w(0xde), w
    ↪(0xdf),\
125     w(0xe0), w(0xe1), w(0xe2), w(0xe3), w(0xe4), w(0xe5), w(0xe6), w
    ↪(0xe7),\
126     w(0xe8), w(0xe9), w(0xea), w(0xeb), w(0xec), w(0xed), w(0xee), w
    ↪(0xef),\
127     w(0xf0), w(0xf1), w(0xf2), w(0xf3), w(0xf4), w(0xf5), w(0xf6), w
    ↪(0xf7),\
128     w(0xf8), w(0xf9), w(0xfa), w(0xfb), w(0xfc), w(0xfd), w(0xfe), w
    ↪(0xff) }
129
130 #define rc_data(w) {\
131     w(0x01), w(0x02), w(0x04), w(0x08), w(0x10),w(0x20), w(0x40), w(0
    ↪x80),\
132     w(0x1b), w(0x36) }
133
134 #define h0(x)    (x)
135
136 #define w0(p)    bytes2word(p, 0, 0, 0)
137 #define w1(p)    bytes2word(0, p, 0, 0)
138 #define w2(p)    bytes2word(0, 0, p, 0)
139 #define w3(p)    bytes2word(0, 0, 0, p)
140
141 #define u0(p)    bytes2word(f2(p), p, p, f3(p))
142 #define u1(p)    bytes2word(f3(p), f2(p), p, p)
143 #define u2(p)    bytes2word(p, f3(p), f2(p), p)
144 #define u3(p)    bytes2word(p, p, f3(p), f2(p))
145
146 #define v0(p)    bytes2word(fe(p), f9(p), fd(p), fb(p))
147 #define v1(p)    bytes2word(fb(p), fe(p), f9(p), fd(p))
148 #define v2(p)    bytes2word(fd(p), fb(p), fe(p), f9(p))
149 #define v3(p)    bytes2word(f9(p), fd(p), fb(p), fe(p))
150
151 #endif
152
153 #if defined(FIXED_TABLES) || !defined(FF_TABLES)
154
155 #define f2(x)    ((x<<1) ^ (((x>>7) & 1) * WPOLY))
156 #define f4(x)    ((x<<2) ^ (((x>>6) & 1) * WPOLY) ^ (((x>>6) & 2) *
    ↪WPOLY))
157 #define f8(x)    ((x<<3) ^ (((x>>5) & 1) * WPOLY) ^ (((x>>5) & 2) *
    ↪WPOLY) \
158                ^ (((x>>5) & 4) * WPOLY))
159 #define f3(x)    (f2(x) ^ x)
160 #define f9(x)    (f8(x) ^ x)
161 #define fb(x)    (f8(x) ^ f2(x) ^ x)
162 #define fd(x)    (f8(x) ^ f4(x) ^ x)
163 #define fe(x)    (f8(x) ^ f4(x) ^ f2(x))
164

```

```

165 #else
166
167 #define f2(x) ((x) ? pow[log[x] + 0x19] : 0)
168 #define f3(x) ((x) ? pow[log[x] + 0x01] : 0)
169 #define f9(x) ((x) ? pow[log[x] + 0xc7] : 0)
170 #define fb(x) ((x) ? pow[log[x] + 0x68] : 0)
171 #define fd(x) ((x) ? pow[log[x] + 0xee] : 0)
172 #define fe(x) ((x) ? pow[log[x] + 0xdf] : 0)
173
174 #endif
175
176 #include "aestab.h"
177
178 #if defined(__cplusplus)
179 extern "C"
180 {
181 #endif
182
183 #if defined(FIXED_TABLES)
184
185 /* implemented in case of wrong call for fixed tables */
186
187 AES_RETURN aes_init(void)
188 {
189     return EXIT_SUCCESS;
190 }
191
192 /* Generate the tables for the dynamic table option */
193
194 #if defined(FF_TABLES)
195
196 #define gf_inv(x) ((x) ? pow[ 255 - log[x]] : 0)
197
198 #else
199
200 /* It will generally be sensible to use tables to compute finite
201 field multiplies and inverses but where memory is scarce this
202 code might sometimes be better. But it only has effect during
203 initialisation so its pretty unimportant in overall terms.
204 */
205
206 /* return 2 ^ (n - 1) where n is the bit number of the highest bit
207 set in x with x in the range 1 < x < 0x00000200. This form is
208 used so that locals within fi can be bytes rather than words
209 */
210
211 static uint_8t hibit(const uint_32t x)
212 {
213     uint_8t r = (uint_8t)((x >> 1) | (x >> 2));
214 }

```

```

214     r |= (r >> 2);
215     r |= (r >> 4);
216     return (r + 1) >> 1;
217 }
218
219 /* return the inverse of the finite field element x */
220
221 static uint_8t gf_inv(const uint_8t x)
222 {
223     uint_8t p1 = x, p2 = BPOLY, n1 = hibit(x), n2 = 0x80, v1 = 1, v2
224     ↪ = 0;
225
226     if(x < 2)
227         return x;
228
229     for( ; ; )
230     {
231         if(n1)
232             while(n2 >= n1) /* divide polynomial p2 by p1
233             ↪ */
234             {
235                 n2 /= n1; /* shift smaller polynomial
236                 ↪ left */
237                 p2 ^= (p1 * n2) & 0xff; /* and remove from larger one
238                 ↪ */
239                 v2 ^= v1 * n2; /* shift accumulated value
240                 ↪ and */
241                 n2 = hibit(p2); /* add into result
242                 ↪ */
243             }
244         else
245             return v1;
246
247         if(n2) /* repeat with values swapped
248         ↪ */
249         while(n1 >= n2)
250         {
251             n1 /= n2;
252             p1 ^= p2 * n1;
253             v1 ^= v2 * n1;
254             n1 = hibit(p1);
255         }
256         else
257             return v2;
258     }
259 }
260
261 #endif
262
263 /* The forward and inverse affine transformations used in the S-box

```

```

↪ */
256 uint_8t fwd_affine(const uint_8t x)
257 {
258     uint_32t w = x;
259     w ^= (w << 1) ^ (w << 2) ^ (w << 3) ^ (w << 4);
260     return 0x63 ^ ((w ^ (w >> 8)) & 0xff);
261 }
262
263 uint_8t inv_affine(const uint_8t x)
264 {
265     uint_32t w = x;
266     w = (w << 1) ^ (w << 3) ^ (w << 6);
267     return 0x05 ^ ((w ^ (w >> 8)) & 0xff);
268 }
269
270 static int init = 0;
271
272 AES_RETURN aes_init(void)
273 {
274     uint_32t i, w;
275
276     #if defined(FF_TABLES)
277
278     uint_8t pow[512], log[256];
279
280     if(init)
281         return EXIT_SUCCESS;
282     /* log and power tables for GF(2^8) finite field with
283        WPOLY as modular polynomial - the simplest primitive
284        root is 0x03, used here to generate the tables
285        */
286
287     i = 0; w = 1;
288     do
289     {
290         pow[i] = (uint_8t)w;
291         pow[i + 255] = (uint_8t)w;
292         log[w] = (uint_8t)i++;
293         w ^= (w << 1) ^ (w & 0x80 ? WPOLY : 0);
294     }
295     while (w != 1);
296
297 #else
298     if(init)
299         return EXIT_SUCCESS;
300 #endif
301
302     for(i = 0, w = 1; i < RC_LENGTH; ++i)
303     {
304         t_set(r,c)[i] = bytes2word(w, 0, 0, 0);
305         w = f2(w);
306     }

```

```

304
305     for(i = 0; i < 256; ++i)
306     {   uint_8t    b;
307
308         b = fwd_affine(gf_inv((uint_8t)i));
309         w = bytes2word(f2(b), b, b, f3(b));
310
311     #if defined( SBX_SET )
312         t_set(s,box)[i] = b;
313     #endif
314
315     #if defined( FT1_SET )           /* tables for a normal
    ↪ encryption round */
316         t_set(f,n)[i] = w;
317     #endif
318     #if defined( FT4_SET )
319         t_set(f,n)[0][i] = w;
320         t_set(f,n)[1][i] = upr(w,1);
321         t_set(f,n)[2][i] = upr(w,2);
322         t_set(f,n)[3][i] = upr(w,3);
323     #endif
324         w = bytes2word(b, 0, 0, 0);
325
326     #if defined( FL1_SET )           /* tables for last encryption round
    ↪ (may also */
327         t_set(f,l)[i] = w;           /* be used in the key schedule)
    ↪ */
328     #endif
329     #if defined( FL4_SET )
330         t_set(f,l)[0][i] = w;
331         t_set(f,l)[1][i] = upr(w,1);
332         t_set(f,l)[2][i] = upr(w,2);
333         t_set(f,l)[3][i] = upr(w,3);
334     #endif
335
336     #if defined( LS1_SET )           /* table for key schedule if
    ↪ t_set(f,l) above is*/
337         t_set(l,s)[i] = w;           /* not of the required form
    ↪ */
338     #endif
339     #if defined( LS4_SET )
340         t_set(l,s)[0][i] = w;
341         t_set(l,s)[1][i] = upr(w,1);
342         t_set(l,s)[2][i] = upr(w,2);
343         t_set(l,s)[3][i] = upr(w,3);
344     #endif
345
346     b = gf_inv(inv_affine((uint_8t)i));
347     w = bytes2word(fe(b), f9(b), fd(b), fb(b));

```

```

348
349 #if defined( IM1_SET )                               /* tables for the inverse mix
350     ↪ column operation */
351     t_set(i,m)[b] = w;
352 #endif
353 #if defined( IM4_SET )
354     t_set(i,m)[0][b] = w;
355     t_set(i,m)[1][b] = upr(w,1);
356     t_set(i,m)[2][b] = upr(w,2);
357     t_set(i,m)[3][b] = upr(w,3);
358 #endif
359 #if defined( ISB_SET )
360     t_set(i,box)[i] = b;
361 #endif
362 #if defined( IT1_SET )                               /* tables for a normal
363     ↪ decryption round */
364     t_set(i,n)[i] = w;
365 #endif
366 #if defined( IT4_SET )
367     t_set(i,n)[0][i] = w;
368     t_set(i,n)[1][i] = upr(w,1);
369     t_set(i,n)[2][i] = upr(w,2);
370     t_set(i,n)[3][i] = upr(w,3);
371 #endif
372     w = bytes2word(b, 0, 0, 0);
373 #if defined( IL1_SET )                               /* tables for last decryption
374     ↪ round */
375     t_set(i,l)[i] = w;
376 #endif
377 #if defined( IL4_SET )
378     t_set(i,l)[0][i] = w;
379     t_set(i,l)[1][i] = upr(w,1);
380     t_set(i,l)[2][i] = upr(w,2);
381     t_set(i,l)[3][i] = upr(w,3);
382 #endif
383     }
384     init = 1;
385     return EXIT_SUCCESS;
386 }
387 #endif
388 #if defined(__cplusplus)
389 }
390 #endif

```

src/Obf/aestab.h

1 */**

```

2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24
25 This file contains the code for declaring the tables needed to
26 ↪implement
27 AES. The file aesopt.h is assumed to be included before this header
28 ↪file.
29 If there are no global variables, the definitions here can be used
30 ↪to put
31 the AES tables in a structure so that a pointer can then be added to
32 ↪the
33 AES context to pass them to the AES routines that need them. If
34 ↪this
35 facility is used, the calling program has to ensure that this
36 ↪pointer is
37 managed appropriately. In particular, the value of the t_dec(in,it)
38 ↪item
39 in the table structure must be set to zero in order to ensure that
40 ↪the
41 tables are initialised. In practice the three code sequences in
42 ↪aeskey.c
43 that control the calls to aes_init() and the aes_init() routine
44 ↪itself will
45 have to be changed for a specific implementation. If global
46 ↪variables are
47 available it will generally be preferable to use them with the
48 ↪precomputed
49 FIXED_TABLES option that uses static global tables.
50
51
52
53

```



```

34  The following defines can be used to control the way the tables
35  are defined, initialised and used in embedded environments that
36  require special features for these purposes
37
38  the 't_dec' construction is used to declare fixed table arrays
39  the 't_set' construction is used to set fixed table values
40  the 't_use' construction is used to access fixed table values
41
42  256 byte tables:
43
44      t_xxx(s,box)    => forward S box
45      t_xxx(i,box)    => inverse S box
46
47  256 32-bit word OR 4 x 256 32-bit word tables:
48
49      t_xxx(f,n)      => forward normal round
50      t_xxx(f,l)      => forward last round
51      t_xxx(i,n)      => inverse normal round
52      t_xxx(i,l)      => inverse last round
53      t_xxx(l,s)      => key schedule table
54      t_xxx(i,m)      => key schedule table
55
56  Other variables and tables:
57
58      t_xxx(r,c)      => the rcon table
59  */
60
61  #if !defined( _AESTAB_H )
62  #define _AESTAB_H
63
64  #if defined(__cplusplus)
65  extern "C" {
66  #endif
67
68  #define t_dec(m,n)  t_###m###n
69  #define t_set(m,n)  t_###m###n
70  #define t_use(m,n)  t_###m###n
71
72  #if defined(FIXED_TABLES)
73  #  if !defined( __GNUC__ ) && (defined( __MSDOS__ ) || defined(
↪ __WIN16__ ))
74  /* make tables far data to avoid using too much DGROUP space (PG)
↪ */
75  #    define CONST const far
76  #  else
77  #    define CONST const
78  #  endif
79  #else
80  #  define CONST

```

```

81 #endif
82
83 #if defined(DO_TABLES)
84 #   define EXTERN
85 #else
86 #   define EXTERN extern
87 #endif
88
89 #if defined(_MSC_VER) && defined(TABLE_ALIGN)
90 #define ALIGN __declspec(align(TABLE_ALIGN))
91 #else
92 #define ALIGN
93 #endif
94
95 #if defined( __WATCOMC__ ) && ( __WATCOMC__ >= 1100 )
96 #   define XP_DIR __cdecl
97 #else
98 #   define XP_DIR
99 #endif
100
101 #if defined(DO_TABLES) && defined(FIXED_TABLES)
102 #define d_1(t,n,b,e)          EXTERN ALIGN CONST XP_DIR t n[256]      =
103                               ↪b(e)
104 #define d_4(t,n,b,e,f,g,h) EXTERN ALIGN CONST XP_DIR t n[4][256] = {
105                               ↪b(e), b(f), b(g), b(h) }
106 EXTERN ALIGN CONST uint_32t t_dec(r,c)[RC_LENGTH] = rc_data(w0);
107 #else
108 #define d_1(t,n,b,e)          EXTERN ALIGN CONST XP_DIR t n[256]
109 #define d_4(t,n,b,e,f,g,h) EXTERN ALIGN CONST XP_DIR t n[4][256]
110 EXTERN ALIGN CONST uint_32t t_dec(r,c)[RC_LENGTH];
111 #endif
112
113 #if defined( SBX_SET )
114     d_1(uint_8t, t_dec(s,box), sb_data, h0);
115 #endif
116
117 #if defined( ISB_SET )
118     d_1(uint_8t, t_dec(i,box), isb_data, h0);
119 #endif
120
121 #if defined( FT1_SET )
122     d_1(uint_32t, t_dec(f,n), sb_data, u0);
123 #endif
124
125 #if defined( FT4_SET )
126     d_4(uint_32t, t_dec(f,n), sb_data, u0, u1, u2, u3);
127 #endif
128
129 #if defined( FL1_SET )
130     d_1(uint_32t, t_dec(f,l), sb_data, w0);
131 #endif

```

```

128 #if defined( FL4_SET )
129     d_4(uint_32t, t_dec(f,l), sb_data, w0, w1, w2, w3);
130 #endif
131
132 #if defined( IT1_SET )
133     d_1(uint_32t, t_dec(i,n), isb_data, v0);
134 #endif
135 #if defined( IT4_SET )
136     d_4(uint_32t, t_dec(i,n), isb_data, v0, v1, v2, v3);
137 #endif
138
139 #if defined( IL1_SET )
140     d_1(uint_32t, t_dec(i,l), isb_data, w0);
141 #endif
142 #if defined( IL4_SET )
143     d_4(uint_32t, t_dec(i,l), isb_data, w0, w1, w2, w3);
144 #endif
145
146 #if defined( LS1_SET )
147 #if defined( FL1_SET )
148 #undef LS1_SET
149 #else
150     d_1(uint_32t, t_dec(l,s), sb_data, w0);
151 #endif
152 #endif
153
154 #if defined( LS4_SET )
155 #if defined( FL4_SET )
156 #undef LS4_SET
157 #else
158     d_4(uint_32t, t_dec(l,s), sb_data, w0, w1, w2, w3);
159 #endif
160 #endif
161
162 #if defined( IM1_SET )
163     d_1(uint_32t, t_dec(i,m), mm_data, v0);
164 #endif
165 #if defined( IM4_SET )
166     d_4(uint_32t, t_dec(i,m), mm_data, v0, v1, v2, v3);
167 #endif
168
169 #if defined(__cplusplus)
170 }
171 #endif
172
173 #endif

```

src/Obf/brg_endian.h

1 /*

```

2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 20/12/2007
24 */
25
26 #ifndef _BRG_ENDIAN_H
27 #define _BRG_ENDIAN_H
28
29 #define IS_BIG_ENDIAN      4321 /* byte 0 is most significant (mc68k)
30 ↪ */
31 #define IS_LITTLE_ENDIAN   1234 /* byte 0 is least significant (i386)
32 ↪ */
33
34 /* Include files where endian defines and byteswap functions may
35 ↪reside */
36 #if defined( __sun )
37 #   include <sys/isa_defs.h>
38 #elif defined( __FreeBSD__ ) || defined( __OpenBSD__ ) || defined(
39 ↪__NetBSD__ )
40 #   include <sys/endian.h>
41 #elif defined( BSD ) && ( BSD >= 199103 ) || defined( __APPLE__ ) ||
42 ↪\
43     defined( __CYGWIN32__ ) || defined( __DJGPP__ ) || defined(
44 ↪__osf__ )
45 #   include <machine/endian.h>
46 #elif defined( __linux__ ) || defined( __GNUC__ ) || defined(
47 ↪__GNU_LIBRARY__ )
48 #   if !defined( __MINGW32__ ) && !defined( _AIX )
49 #       include <endian.h>
50 #   if !defined( __BEOS__ )

```

```

39 #         include <byteswap.h>
40 #     endif
41 # endif
42 #endif
43
44 /* Now attempt to set the define for platform byte order using any
↳ */
45 /* of the four forms SYMBOL, _SYMBOL, __SYMBOL & __SYMBOL__, which
↳ */
46 /* seem to encompass most endian symbol definitions
↳ */
47
48 #if defined( BIG_ENDIAN ) && defined( LITTLE_ENDIAN )
49 #   if defined( BYTE_ORDER ) && BYTE_ORDER == BIG_ENDIAN
50 #       define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
51 #   elif defined( BYTE_ORDER ) && BYTE_ORDER == LITTLE_ENDIAN
52 #       define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
53 #   endif
54 #elif defined( BIG_ENDIAN )
55 #   define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
56 #elif defined( LITTLE_ENDIAN )
57 #   define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
58 #endif
59
60 #if defined( _BIG_ENDIAN ) && defined( _LITTLE_ENDIAN )
61 #   if defined( _BYTE_ORDER ) && _BYTE_ORDER == _BIG_ENDIAN
62 #       define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
63 #   elif defined( _BYTE_ORDER ) && _BYTE_ORDER == _LITTLE_ENDIAN
64 #       define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
65 #   endif
66 #elif defined( _BIG_ENDIAN )
67 #   define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
68 #elif defined( _LITTLE_ENDIAN )
69 #   define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
70 #endif
71
72 #if defined( __BIG_ENDIAN ) && defined( __LITTLE_ENDIAN )
73 #   if defined( __BYTE_ORDER ) && __BYTE_ORDER == __BIG_ENDIAN
74 #       define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
75 #   elif defined( __BYTE_ORDER ) && __BYTE_ORDER == __LITTLE_ENDIAN
76 #       define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
77 #   endif
78 #elif defined( __BIG_ENDIAN )
79 #   define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
80 #elif defined( __LITTLE_ENDIAN )
81 #   define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
82 #endif
83
84 #if defined( __BIG_ENDIAN__ ) && defined( __LITTLE_ENDIAN__ )

```

```

85 # if defined( __BYTE_ORDER__ ) && __BYTE_ORDER__ == __BIG_ENDIAN__
86 #     define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
87 # elif defined( __BYTE_ORDER__ ) && __BYTE_ORDER__ ==
    ↪ __LITTLE_ENDIAN__
88 #     define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
89 # endif
90 #elif defined( __BIG_ENDIAN__ )
91 #     define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
92 #elif defined( __LITTLE_ENDIAN__ )
93 #     define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
94 #endif
95
96 /* if the platform byte order could not be determined, then try to
    ↪ */
97 /* set this define using common machine defines
    ↪ */
98 #if !defined(PLATFORM_BYTE_ORDER)
99
100 #if defined( __alpha__ ) || defined( __alpha ) || defined( i386 )
    ↪ || \
101     defined( __i386__ ) || defined( _M_I86 ) || defined( _M_IX86
    ↪ ) || \
102     defined( __OS2__ ) || defined( sun386 ) || defined(
    ↪ __TURBOC__ ) || \
103     defined( vax ) || defined( vms ) || defined( VMS )
    ↪ || \
104     defined( __VMS ) || defined( _M_X64 )
105 # define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
106
107 #elif defined( AMIGA ) || defined( applec ) || defined(
    ↪ __AS400__ ) || \
108     defined( _CRAY ) || defined( __hppa ) || defined( __hp9000
    ↪ ) || \
109     defined( ibm370 ) || defined( mc68000 ) || defined( m68k )
    ↪ || \
110     defined( __MRC__ ) || defined( __MVS__ ) || defined(
    ↪ __MWERKS__ ) || \
111     defined( sparc ) || defined( __sparc ) || defined(
    ↪ SYMANTEC_C ) || \
112     defined( __VOS__ ) || defined( __TIGCC__ ) || defined( __TANDEM
    ↪ ) || \
113     defined( THINK_C ) || defined( __VMCMS__ ) || defined( _AIX )
114 # define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
115
116 #elif 0 /* **** EDIT HERE IF NECESSARY **** */
117 # define PLATFORM_BYTE_ORDER IS_LITTLE_ENDIAN
118 #elif 0 /* **** EDIT HERE IF NECESSARY **** */
119 # define PLATFORM_BYTE_ORDER IS_BIG_ENDIAN
120 #else

```

```

121 # error Please edit lines 126 or 128 in brg_endian.h to set the
    ↳platform byte order
122 #endif
123
124 #endif
125
126 #endif

src/Obf/brg_types.h

1  /*
2  -----
3  ↳
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↳reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↳warranties
18 in respect of its operation, including, but not limited to,
19 ↳correctness
20 and fitness for purpose.
21 -----
22 ↳
23 Issue Date: 20/12/2007
24
25 The unsigned integer types defined here are of the form uint_<nn>t
26 ↳where
27 <nn> is the length of the type; for example, the unsigned 32-bit
28 ↳type is
29 'uint_32t'. These are NOT the same as the 'C99 integer types' that
30 ↳are
31 defined in the inttypes.h and stdint.h headers since attempts to use
32 ↳these
33 types have shown that support for them is still highly variable.
34 ↳However,
35 since the latter are of the form uint<nn>_t, a regular expression
36 ↳search
37 and replace (in VC++ search on 'uint_{:x}t' and replace with 'uint\1
38 ↳_t')
39 can be used to convert the types used here to the C99 standard types
40 ↳.

```

```

28  */
29
30 #ifndef _BRG_TYPES_H
31 #define _BRG_TYPES_H
32
33 #if defined(__cplusplus)
34 extern "C" {
35 #endif
36
37 #include <limits.h>
38
39 #if defined( _MSC_VER ) && ( _MSC_VER >= 1300 )
40 # include <stddef.h>
41 # define ptrint_t intptr_t
42 #elif defined( __ECOS__ )
43 # define intptr_t unsigned int
44 # define ptrint_t intptr_t
45 #elif defined( __GNUC__ ) && ( __GNUC__ >= 3 )
46 # include <stdint.h>
47 # define ptrint_t intptr_t
48 #else
49 # define ptrint_t int
50 #endif
51
52 #ifndef BRG_UI8
53 # define BRG_UI8
54 # if UCHAR_MAX == 255u
55     typedef unsigned char uint_8t;
56 # else
57 #   error Please define uint_8t as an 8-bit unsigned integer type in
58   ↪ brg_types.h
59 # endif
60 #endif
61
62 #ifndef BRG_UI16
63 # define BRG_UI16
64 # if USHRT_MAX == 65535u
65     typedef unsigned short uint_16t;
66 # else
67 #   error Please define uint_16t as a 16-bit unsigned short type in
68   ↪ brg_types.h
69 # endif
70 #endif
71
72 #ifndef BRG_UI32
73 # define BRG_UI32
74 # if UINT_MAX == 4294967295u
75     define li_32(h) 0x##h##u
76     typedef unsigned int uint_32t;

```



```

75 # elif ULONG_MAX == 4294967295u
76 #     define li_32(h) 0x##h##ul
77     typedef unsigned long uint_32t;
78 # elif defined( _CRAY )
79 #     error This code needs 32-bit data types, which Cray machines do
    ↪not provide
80 # else
81 #     error Please define uint_32t as a 32-bit unsigned integer type
    ↪in brg_types.h
82 # endif
83 #endif
84
85 #ifndef BRG_UI64
86 # if defined( __BORLANDC__ ) && !defined( __MSDOS__ )
87 #     define BRG_UI64
88 #     define li_64(h) 0x##h##ui64
89     typedef unsigned __int64 uint_64t;
90 # elif defined( _MSC_VER ) && ( _MSC_VER < 1300 )    /* 1300 == VC++
    ↪ 7.0 */
91 #     define BRG_UI64
92 #     define li_64(h) 0x##h##ui64
93     typedef unsigned __int64 uint_64t;
94 # elif defined( __sun ) && defined( ULONG_MAX ) && ULONG_MAX == 0
    ↪xfffffffful
95 #     define BRG_UI64
96 #     define li_64(h) 0x##h##ull
97     typedef unsigned long long uint_64t;
98 # elif defined( __MVS__ )
99 #     define BRG_UI64
100 #     define li_64(h) 0x##h##ull
101     typedef unsigned int long long uint_64t;
102 # elif defined( UINT_MAX ) && UINT_MAX > 4294967295u
103 #     if UINT_MAX == 18446744073709551615u
104 #         define BRG_UI64
105 #         define li_64(h) 0x##h##u
106         typedef unsigned int uint_64t;
107 #     endif
108 # elif defined( ULONG_MAX ) && ULONG_MAX > 4294967295u
109 #     if ULONG_MAX == 18446744073709551615ul
110 #         define BRG_UI64
111 #         define li_64(h) 0x##h##ul
112         typedef unsigned long uint_64t;
113 #     endif
114 # elif defined( ULLONG_MAX ) && ULLONG_MAX > 4294967295u
115 #     if ULLONG_MAX == 18446744073709551615ull
116 #         define BRG_UI64
117 #         define li_64(h) 0x##h##ull
118         typedef unsigned long long uint_64t;
119 #     endif

```

```

120 # elif defined( ULONG_LONG_MAX ) && ULONG_LONG_MAX > 4294967295u
121 #     if ULONG_LONG_MAX == 18446744073709551615ull
122 #         define BRG_UI64
123 #         define li_64(h) 0x##h##ull
124 #         typedef unsigned long long uint_64t;
125 #     endif
126 # endif
127 #endif
128
129 #if !defined( BRG_UI64 )
130 # if defined( NEED_UINT_64T )
131 #     error Please define uint_64t as an unsigned 64 bit type in
132 ↪brg_types.h
133 # endif
134 #endif
135
136 #ifndef RETURN_VALUES
137 # define RETURN_VALUES
138 # if defined( DLL_EXPORT )
139 #     if defined( _MSC_VER ) || defined( __INTEL_COMPILER )
140 #         define VOID_RETURN __declspec( dllexport ) void __stdcall
141 #         define INT_RETURN __declspec( dllexport ) int __stdcall
142 #     elif defined( __GNUC__ )
143 #         define VOID_RETURN __declspec( __dllexport__ ) void
144 #         define INT_RETURN __declspec( __dllexport__ ) int
145 #     else
146 #         error Use of the DLL is only available on the Microsoft, Intel
147 ↪and GCC compilers
148 #     endif
149 # elif defined( DLL_IMPORT )
150 #     if defined( _MSC_VER ) || defined( __INTEL_COMPILER )
151 #         define VOID_RETURN __declspec( dllimport ) void __stdcall
152 #         define INT_RETURN __declspec( dllimport ) int __stdcall
153 #     elif defined( __GNUC__ )
154 #         define VOID_RETURN __declspec( __dllimport__ ) void
155 #         define INT_RETURN __declspec( __dllimport__ ) int
156 #     else
157 #         error Use of the DLL is only available on the Microsoft, Intel
158 ↪and GCC compilers
159 #     endif
160 # elif defined( __WATCOMC__ )
161 #     define VOID_RETURN void __cdecl
162 #     define INT_RETURN int __cdecl
163 # else
164 #     define VOID_RETURN void
165 #     define INT_RETURN int
166 # endif
167 #endif

```

```

166  /*      These defines are used to detect and set the memory alignment
      ↳ of pointers.
167      Note that offsets are in bytes.
168
169      ALIGN_OFFSET(x,n)          return the positive or zero
      ↳offset of                  the memory addressed by the pointer '
170      ↳x'                        from an address that is aligned on an
171                                'n' byte boundary ('n' is a power of
172      ↳2)
173
174      ALIGN_FLOOR(x,n)          return a pointer that points
      ↳to memory                  that is aligned on an 'n' byte
175                                ↳boundary
176                                and is not higher than the memory
      ↳address                    pointed to by 'x' ('n' is a power of
177      ↳2)
178
179      ALIGN_CEIL(x,n)           return a pointer that
      ↳points to memory           that is aligned on an 'n' byte
180                                ↳boundary
181                                and is not lower than the memory
      ↳address                    pointed to by 'x' ('n' is a power of
182      ↳2)
183  */
184
185  #define ALIGN_OFFSET(x,n)      (((puint_t)(x)) & ((n) - 1))
186  #define ALIGN_FLOOR(x,n)      ((uint_8t*)(x) - (((puint_t)(x)) &
      ↳((n) - 1)))
187  #define ALIGN_CEIL(x,n)       ((uint_8t*)(x) + (-((puint_t)(x)) &
      ↳((n) - 1)))
188
189  /*      These defines are used to declare buffers in a way that allows
190      faster operations on longer variables to be used. In all these
191      defines 'size' must be a power of 2 and >= 8. NOTE that the
192      buffer size is in bytes but the type length is in bits
193
194      UNIT_TYPEDEF(x,size)      declares a variable 'x' of length
195                                'size' bits
196
197      BUFR_TYPEDEF(x,size,bsize) declares a buffer 'x' of length '
      ↳bsize'
198                                bytes defined as an array of
      ↳variables

```

```

199                                     each of 'size' bits (bsize must be a
200                                     multiple of size / 8)
201
202     UNIT_CAST(x,size)               casts a variable to a type of
203                                     length 'size' bits
204
205     UPTR_CAST(x,size)              casts a pointer to a pointer to a
206                                     variable of length 'size' bits
207 */
208
209 #define UI_TYPE(size)               uint_##size##t
210 #define UNIT_TYPEDEF(x,size)       typedef UI_TYPE(size) x
211 #define BUFR_TYPEDEF(x,size,bsize) typedef UI_TYPE(size) x[bsize / (
    ↪size >> 3)]
212 #define UNIT_CAST(x,size)          ((UI_TYPE(size) )(x))
213 #define UPTR_CAST(x,size)          ((UI_TYPE(size)*) (x))
214
215 #if defined(__cplusplus)
216 }
217 #endif
218
219 #endif

```

src/Obf/cmac.c

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 6/10/2008
24 */
25

```

```

21 #include "cmac.h"
22
23 #define BLK_ADR_MASK      (BLOCK_SIZE - 1)
24
25 void cmac_init(const unsigned char key[], cmac_ctx ctx[1])
26 {
27     memset(ctx, 0, sizeof(cmac_ctx));
28     aes_encrypt_key128(key, ctx->aes);
29 }
30
31 void cmac_data(const unsigned char buf[], unsigned long len, cmac_ctx
32   ↪ ctx[1])
33 {
34     uint_32t cnt = 0, b_pos = ctx->txt_cnt & BLK_ADR_MASK;
35
36     if(!len)
37         return;
38
39     if(!((buf - (UI8_PTR(ctx->txt_cbc) + b_pos)) & BUF_ADRMASK))
40     {
41         while(cnt < len && (b_pos & BUF_ADRMASK))
42             UI8_PTR(ctx->txt_cbc)[b_pos++] ^= buf[cnt++];
43
44         while(cnt + BLOCK_SIZE <= len)
45         {
46             while(cnt + BUF_INC <= len && b_pos <= BLOCK_SIZE -
47   ↪ BUF_INC)
48             {
49                 *UNIT_PTR(UI8_PTR(ctx->txt_cbc) + b_pos) ^= *UNIT_PTR
50   ↪ (buf + cnt);
51                 cnt += BUF_INC; b_pos += BUF_INC;
52             }
53             while(cnt + BLOCK_SIZE <= len)
54             {
55                 aes_ecb_encrypt(UI8_PTR(ctx->txt_cbc), UI8_PTR(ctx->
56   ↪ txt_cbc), AES_BLOCK_SIZE, ctx->aes);
57                 xor_block_aligned(ctx->txt_cbc, ctx->txt_cbc, buf +
58   ↪ cnt);
59                 cnt += BLOCK_SIZE;
60             }
61         }
62     }
63     else
64     {
65         while(cnt < len && b_pos < BLOCK_SIZE)
66             UI8_PTR(ctx->txt_cbc)[b_pos++] ^= buf[cnt++];
67
68         while(cnt + BLOCK_SIZE <= len)
69         {

```

```

65         aes_ecb_encrypt(UI8_PTR(ctx->txt_cbc), UI8_PTR(ctx->
↪txt_cbc), AES_BLOCK_SIZE, ctx->aes);
66         xor_block(ctx->txt_cbc, ctx->txt_cbc, buf + cnt);
67         cnt += BLOCK_SIZE;
68     }
69 }
70
71 while(cnt < len)
72 {
73     if(b_pos == BLOCK_SIZE)
74     {
75         aes_ecb_encrypt(UI8_PTR(ctx->txt_cbc), UI8_PTR(ctx->
↪txt_cbc), AES_BLOCK_SIZE, ctx->aes);
76         b_pos = 0;
77     }
78     UI8_PTR(ctx->txt_cbc)[b_pos++] ^= buf[cnt++];
79 }
80
81 ctx->txt_cnt += cnt;
82 }
83
84 static const unsigned char c_xor[4] = { 0x00, 0x87, 0x0e, 0x89 };
85
86 static void gf_mulx(uint_8t pad[BLOCK_SIZE])
87 {     int i, t = pad[0] >> 7;
88
89     for(i = 0; i < BLOCK_SIZE - 1; ++i)
90         pad[i] = (pad[i] << 1) | (pad[i + 1] >> 7);
91     pad[BLOCK_SIZE - 1] = (pad[BLOCK_SIZE - 1] << 1) ^ c_xor[t];
92 }
93
94 void gf_mulx2(uint_8t pad[BLOCK_SIZE])
95 {     int i, t = pad[0] >> 6;
96
97     for(i = 0; i < BLOCK_SIZE - 1; ++i)
98         pad[i] = (pad[i] << 2) | (pad[i + 1] >> 6);
99     pad[BLOCK_SIZE - 2] ^= (t >> 1);
100    pad[BLOCK_SIZE - 1] = (pad[BLOCK_SIZE - 1] << 2) ^ c_xor[t];
101 }
102
103 void cmac_end(unsigned char auth_tag[], cmac_ctx ctx[1])
104 {     buf_type pad;
105     int i;
106
107     memset(pad, 0, sizeof(pad));
108     aes_ecb_encrypt(UI8_PTR(pad), UI8_PTR(pad), AES_BLOCK_SIZE, ctx->
↪aes);
109     i = ctx->txt_cnt & BLK_ADR_MASK;
110     if(ctx->txt_cnt == 0 || i)

```

```

111     {
112         UI8_PTR(ctx->txt_cbc)[i] ^= 0x80;
113         gf_mulx2(UI8_PTR(pad));
114     }
115     else
116         gf_mulx(UI8_PTR(pad));
117
118     xor_block_aligned(pad, pad, ctx->txt_cbc);
119     aes_ecb_encrypt(UI8_PTR(pad), UI8_PTR(pad), AES_BLOCK_SIZE, ctx->
120 ↪aes);
121
122     for(i = 0; i < BLOCK_SIZE; ++i)
123         auth_tag[i] = UI8_PTR(pad)[i];
124 }

```

src/Obf/cmac.h

```

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 6/10/2008
24 */
25
26 #ifndef CMAC_AES_H
27 #define CMAC_AES_H
28
29 #if !defined( UNIT_BITS )
30 #   if 1
31 #       define UNIT_BITS 64
32 #   elif 0
33 #       define UNIT_BITS 32

```

```

29 # else
30 #     define UNIT_BITS 8
31 # endif
32 #endif
33
34 #include <string.h>
35 #include "aes.h"
36 #include "mode_hdr.h"
37
38 UNIT_TYPEDEF(buf_unit, UNIT_BITS);
39 BUFR_TYPEDEF(buf_type, UNIT_BITS, AES_BLOCK_SIZE);
40
41 #if defined(__cplusplus)
42 extern "C"
43 {
44 #endif
45
46 #define BLOCK_SIZE  AES_BLOCK_SIZE
47
48 typedef struct
49 {
50     buf_type          txt_cbc;
51     aes_encrypt_ctx  aes[1];           /* AES encryption context
↪      */
52     uint_32t         txt_cnt;
53 } cmac_ctx;
54
55 void cmac_init( const unsigned char key[], /* the encryption key
↪      */
56                cmac_ctx ctx[1] );        /* the OMAC context
↪      */
57
58 void cmac_data( const unsigned char buf[], /* the data buffer
↪      */
59                unsigned long len,         /* the length of this
↪ block (bytes) */
60                cmac_ctx ctx[1] );        /* the OMAC context
↪      */
61
62 void cmac_end( unsigned char auth_tag[], /* the encryption key
↪      */
63               cmac_ctx ctx[1] );        /* the OMAC context
↪      */
64
65 #if defined(__cplusplus)
66 }
67 #endif
68
69 #endif

```



```

src/Obf/mode_hdr.h

1  /*
2  -----
3  ↪
4  Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights
5  ↪reserved.
6
7  The redistribution and use of this software (with or without changes)
8  is allowed without the payment of fees or royalties provided that:
9
10     source code distributions include the above copyright notice, this
11     list of conditions and the following disclaimer;
12
13     binary distributions include the above copyright notice, this list
14     of conditions and the following disclaimer in their documentation.
15
16 This software is provided 'as is' with no explicit or implied
17 ↪warranties
18 in respect of its operation, including, but not limited to,
19 ↪correctness
20 and fitness for purpose.
21 -----
22 ↪
23 Issue Date: 07/10/2010
24
25 This header file is an INTERNAL file which supports mode
26 ↪implementation
27 */
28
29 #ifndef _MODE_HDR_H
30 #define _MODE_HDR_H
31
32 #include <string.h>
33 #include <limits.h>
34
35 #include "brg_endian.h"
36
37 /* This define sets the units in which buffers are processed. This
38 ↪code
39     can provide significant speed gains if buffers can be processed
40 ↪in
41     32 or 64 bit chunks rather than in bytes. This define sets the
42 ↪units
43     in which buffers will be accessed if possible
44 */
45 #if !defined( UNIT_BITS )
46 #   if PLATFORM_BYTE_ORDER == IS_BIG_ENDIAN
47 #       if 0
48 #           define UNIT_BITS    32

```

```

40 #     elif 1
41 #         define UNIT_BITS 64
42 #     endif
43 # elif defined( _WIN64 )
44 #     define UNIT_BITS 64
45 # else
46 #     define UNIT_BITS 32
47 # endif
48 #endif
49
50 #if UNIT_BITS == 64 && !defined( NEED_UINT_64T )
51 # define NEED_UINT_64T
52 #endif
53
54 #include "brg_types.h"
55
56 /* Use of inlines is preferred but code blocks can also be expanded
57 ↪ inline
58     using 'defines'. But the latter approach will typically generate
59 ↪ a LOT
60     of code and is not recommended.
61 */
62 #if 1 && !defined( USE_INLINING )
63 # define USE_INLINING
64 #endif
65
66 #if defined( _MSC_VER )
67 # if _MSC_VER >= 1400
68 #     include <stdlib.h>
69 #     include <intrin.h>
70 #     pragma intrinsic(memset)
71 #     pragma intrinsic(memcpy)
72 #     define rotl32      _rotl
73 #     define rotr32      _rotr
74 #     define rotl64      _rotl64
75 #     define rotr64      _rotr64
76 #     define bswap_16(x) _byteswap_ushort(x)
77 #     define bswap_32(x) _byteswap_ulong(x)
78 #     define bswap_64(x) _byteswap_uint64(x)
79 # else
80 #     define rotl32 _lrotl
81 #     define rotr32 _lrotr
82 # endif
83 #endif
84
85 #if defined( USE_INLINING )
86 # if defined( _MSC_VER )
87 #     define mh_decl __inline
88 # elif defined( __GNUC__ ) || defined( __GNU_LIBRARY__ )

```

```

87 #   define mh_decl static inline
88 #   else
89 #   define mh_decl static
90 #   endif
91 #endif
92
93 #if defined(__cplusplus)
94 extern "C" {
95 #endif
96
97 #define UI8_PTR(x)      UPTR_CAST(x, 8)
98 #define UI16_PTR(x)     UPTR_CAST(x, 16)
99 #define UI32_PTR(x)     UPTR_CAST(x, 32)
100 #define UI64_PTR(x)     UPTR_CAST(x, 64)
101 #define UNIT_PTR(x)     UPTR_CAST(x, UNIT_BITS)
102
103 #define UI8_VAL(x)      UNIT_CAST(x, 8)
104 #define UI16_VAL(x)     UNIT_CAST(x, 16)
105 #define UI32_VAL(x)     UNIT_CAST(x, 32)
106 #define UI64_VAL(x)     UNIT_CAST(x, 64)
107 #define UNIT_VAL(x)     UNIT_CAST(x, UNIT_BITS)
108
109 #define BUF_INC          (UNIT_BITS >> 3)
110 #define BUF_ADRMASK      ((UNIT_BITS >> 3) - 1)
111
112 #define rep2_u2(f,r,x)    f( 0,r,x); f( 1,r,x)
113 #define rep2_u4(f,r,x)    f( 0,r,x); f( 1,r,x); f( 2,r,x); f( 3,r,x)
114 #define rep2_u16(f,r,x)   f( 0,r,x); f( 1,r,x); f( 2,r,x); f( 3,r,x);
115 ↪ \
116                               f( 4,r,x); f( 5,r,x); f( 6,r,x); f( 7,r,x);
117 ↪ \
118                               f( 8,r,x); f( 9,r,x); f(10,r,x); f(11,r,x);
119 ↪ \
120                               f(12,r,x); f(13,r,x); f(14,r,x); f(15,r,x)
121
122 #define rep2_d2(f,r,x)    f( 1,r,x); f( 0,r,x)
123 #define rep2_d4(f,r,x)    f( 3,r,x); f( 2,r,x); f( 1,r,x); f( 0,r,x)
124 #define rep2_d16(f,r,x)   f(15,r,x); f(14,r,x); f(13,r,x); f(12,r,x);
125 ↪ \
126                               f(11,r,x); f(10,r,x); f( 9,r,x); f( 8,r,x);
127 ↪ \
128                               f( 7,r,x); f( 6,r,x); f( 5,r,x); f( 4,r,x);
129 ↪ \
130                               f( 3,r,x); f( 2,r,x); f( 1,r,x); f( 0,r,x)
131
132 #define rep3_u2(f,r,x,y,c) f( 0,r,x,y,c); f( 1,r,x,y,c)
133 #define rep3_u4(f,r,x,y,c) f( 0,r,x,y,c); f( 1,r,x,y,c); f( 2,r,x,y,
134 ↪ c); f( 3,r,x,y,c)
135 #define rep3_u16(f,r,x,y,c) f( 0,r,x,y,c); f( 1,r,x,y,c); f( 2,r,x,y,

```

```

129     ↪c); f( 3,r,x,y,c); \
130     ↪c); f( 7,r,x,y,c); \
131     ↪c); f(11,r,x,y,c); \
132     ↪c); f(15,r,x,y,c)
133     #define rep3_d2(f,r,x,y,c) f( 1,r,x,y,c); f( 0,r,x,y,c)
134     #define rep3_d4(f,r,x,y,c) f( 3,r,x,y,c); f( 2,r,x,y,c); f( 1,r,x,y,
135     ↪c); f( 0,r,x,y,c)
136     #define rep3_d16(f,r,x,y,c) f(15,r,x,y,c); f(14,r,x,y,c); f(13,r,x,y,
137     ↪c); f(12,r,x,y,c); \
138     ↪c); f( 8,r,x,y,c); \
139     ↪c); f( 4,r,x,y,c); \
140     ↪c); f( 0,r,x,y,c)
141     f( 4,r,x,y,c); f( 5,r,x,y,c); f( 6,r,x,y,
142     f( 8,r,x,y,c); f( 9,r,x,y,c); f(10,r,x,y,
143     f(12,r,x,y,c); f(13,r,x,y,c); f(14,r,x,y,
144     f(11,r,x,y,c); f(10,r,x,y,c); f( 9,r,x,y,
145     f( 7,r,x,y,c); f( 6,r,x,y,c); f( 5,r,x,y,
146     f( 3,r,x,y,c); f( 2,r,x,y,c); f( 1,r,x,y,
147     /* function pointers might be used for fast XOR operations */
148     typedef void (*xor_function)(void* r, const void* p, const void* q);
149     /* left and right rotates on 32 and 64 bit variables */
150     #if !defined( rotl32 ) /* NOTE: 0 <= n <= 32 ASSUMED */
151     mh_decl uint_32t rotl32(uint_32t x, int n)
152     {
153         return (((x) << n) | ((x) >> (32 - n)));
154     }
155     #endif
156     #if !defined( rotr32 ) /* NOTE: 0 <= n <= 32 ASSUMED */
157     mh_decl uint_32t rotr32(uint_32t x, int n)
158     {
159         return (((x) >> n) | ((x) << (32 - n)));
160     }
161     #endif
162     #if UNIT_BITS == 64 && !defined( rotl64 ) /* NOTE: 0 <= n <= 64
163     ↪ASSUMED */
164     mh_decl uint_64t rotl64(uint_64t x, int n)
165     {
166         return (((x) << n) | ((x) >> (64 - n)));
167     }
168     #endif
169     #if UNIT_BITS == 64 && !defined( rotr64 ) /* NOTE: 0 <= n <= 64

```

```

↪ASSUMED */
168 mh_decl uint_64t rotr64(uint_64t x, int n)
169 {
170     return (((x) >> n) | ((x) << (64 - n)));
171 }
172 #endif
173
174 /* byte order inversions for 16, 32 and 64 bit variables */
175
176 #if !defined(bswap_16)
177 mh_decl uint_16t bswap_16(uint_16t x)
178 {
179     return (uint_16t)((x >> 8) | (x << 8));
180 }
181 #endif
182
183 #if !defined(bswap_32)
184 mh_decl uint_32t bswap_32(uint_32t x)
185 {
186     return ((rotr32((x), 24) & 0x00ff00ff) | (rotr32((x), 8) & 0
↪x00ff00ff));
187 }
188 #endif
189
190 #if UNIT_BITS == 64 && !defined(bswap_64)
191 mh_decl uint_64t bswap_64(uint_64t x)
192 {
193     return bswap_32((uint_32t)(x >> 32)) | ((uint_64t)bswap_32(
↪uint_32t)x) << 32);
194 }
195 #endif
196
197 /* support for fast aligned buffer move, xor and byte swap operations
↪ -
198     source and destination buffers for move and xor operations must
↪not
199     overlap, those for byte order reversal must either not overlap or
200     must be identical
201 */
202 #define f_copy(n,p,q)    p[n] = q[n]
203 #define f_xor(n,r,p,q,c) r[n] = c(p[n] ^ q[n])
204
205 mh_decl void copy_block(void* p, const void* q)
206 {
207     memcpy(p, q, 16);
208 }
209
210 mh_decl void copy_block_aligned(void *p, const void *q)
211 {

```

```

212 #if UNIT_BITS == 8
213     memcpy(p, q, 16);
214 #elif UNIT_BITS == 32
215     rep2_u4(f_copy, UNIT_PTR(p), UNIT_PTR(q));
216 #else
217     rep2_u2(f_copy, UNIT_PTR(p), UNIT_PTR(q));
218 #endif
219 }
220
221 mh_decl void xor_block(void *r, const void* p, const void* q)
222 {
223     rep3_u16(f_xor, UI8_PTR(r), UI8_PTR(p), UI8_PTR(q), UI8_VAL);
224 }
225
226 mh_decl void xor_block_aligned(void *r, const void *p, const void *q)
227 {
228     #if UNIT_BITS == 8
229         rep3_u16(f_xor, UNIT_PTR(r), UNIT_PTR(p), UNIT_PTR(q), UNIT_VAL);
230     #elif UNIT_BITS == 32
231         rep3_u4(f_xor, UNIT_PTR(r), UNIT_PTR(p), UNIT_PTR(q), UNIT_VAL);
232     #else
233         rep3_u2(f_xor, UNIT_PTR(r), UNIT_PTR(p), UNIT_PTR(q), UNIT_VAL);
234     #endif
235 }
236
237 /* byte swap within 32-bit words in a 16 byte block; don't move 32-
238    ↪bit words */
239 mh_decl void bswap32_block(void *d, const void* s)
240 {
241     #if UNIT_BITS == 8
242         uint_8t t;
243         t = UNIT_PTR(s)[ 0]; UNIT_PTR(d)[ 0] = UNIT_PTR(s)[ 3]; UNIT_PTR(
244 ↪d)[ 3] = t;
245         t = UNIT_PTR(s)[ 1]; UNIT_PTR(d)[ 1] = UNIT_PTR(s)[ 2]; UNIT_PTR(
246 ↪d)[ 2] = t;
247         t = UNIT_PTR(s)[ 4]; UNIT_PTR(d)[ 4] = UNIT_PTR(s)[ 7]; UNIT_PTR(
248 ↪d)[ 7] = t;
249         t = UNIT_PTR(s)[ 5]; UNIT_PTR(d)[ 5] = UNIT_PTR(s)[ 6]; UNIT_PTR(
250 ↪d)[ 6] = t;
251         t = UNIT_PTR(s)[ 8]; UNIT_PTR(d)[ 8] = UNIT_PTR(s)[11]; UNIT_PTR(
252 ↪d)[12] = t;
253         t = UNIT_PTR(s)[ 9]; UNIT_PTR(d)[ 9] = UNIT_PTR(s)[10]; UNIT_PTR(
254 ↪d)[10] = t;
255         t = UNIT_PTR(s)[12]; UNIT_PTR(d)[12] = UNIT_PTR(s)[15]; UNIT_PTR(
256 ↪d)[15] = t;
257         t = UNIT_PTR(s)[13]; UNIT_PTR(d)[ 3] = UNIT_PTR(s)[14]; UNIT_PTR(
258 ↪d)[14] = t;
259     #elif UNIT_BITS == 32
260         UNIT_PTR(d)[0] = bswap_32(UNIT_PTR(s)[0]); UNIT_PTR(d)[1] =

```

```

↪bswap_32(UNIT_PTR(s)[1]);
252     UNIT_PTR(d)[2] = bswap_32(UNIT_PTR(s)[2]); UNIT_PTR(d)[3] =
↪bswap_32(UNIT_PTR(s)[3]);
253 #else
254     UI32_PTR(d)[0] = bswap_32(UI32_PTR(s)[0]); UI32_PTR(d)[1] =
↪bswap_32(UI32_PTR(s)[1]);
255     UI32_PTR(d)[2] = bswap_32(UI32_PTR(s)[2]); UI32_PTR(d)[3] =
↪bswap_32(UI32_PTR(s)[3]);
256 #endif
257 }
258
259 /* byte swap within 64-bit words in a 16 byte block; don't move 64-
↪bit words */
260 mh_decl void bswap64_block(void *d, const void* s)
261 {
262     #if UNIT_BITS == 8
263         uint_8t t;
264         t = UNIT_PTR(s)[ 0]; UNIT_PTR(d)[ 0] = UNIT_PTR(s)[ 7]; UNIT_PTR(
↪d)[ 7] = t;
265         t = UNIT_PTR(s)[ 1]; UNIT_PTR(d)[ 1] = UNIT_PTR(s)[ 6]; UNIT_PTR(
↪d)[ 6] = t;
266         t = UNIT_PTR(s)[ 2]; UNIT_PTR(d)[ 2] = UNIT_PTR(s)[ 5]; UNIT_PTR(
↪d)[ 5] = t;
267         t = UNIT_PTR(s)[ 3]; UNIT_PTR(d)[ 3] = UNIT_PTR(s)[ 4]; UNIT_PTR(
↪d)[ 4] = t;
268         t = UNIT_PTR(s)[ 8]; UNIT_PTR(d)[ 8] = UNIT_PTR(s)[15]; UNIT_PTR(
↪d)[15] = t;
269         t = UNIT_PTR(s)[ 9]; UNIT_PTR(d)[ 9] = UNIT_PTR(s)[14]; UNIT_PTR(
↪d)[14] = t;
270         t = UNIT_PTR(s)[10]; UNIT_PTR(d)[10] = UNIT_PTR(s)[13]; UNIT_PTR(
↪d)[13] = t;
271         t = UNIT_PTR(s)[11]; UNIT_PTR(d)[11] = UNIT_PTR(s)[12]; UNIT_PTR(
↪d)[12] = t;
272     #elif UNIT_BITS == 32
273         uint_32t t;
274         t = bswap_32(UNIT_PTR(s)[0]); UNIT_PTR(d)[0] = bswap_32(UNIT_PTR(
↪s)[1]); UNIT_PTR(d)[1] = t;
275         t = bswap_32(UNIT_PTR(s)[2]); UNIT_PTR(d)[2] = bswap_32(UNIT_PTR(
↪s)[3]); UNIT_PTR(d)[3] = t;
276     #else
277         UNIT_PTR(d)[0] = bswap_64(UNIT_PTR(s)[0]); UNIT_PTR(d)[1] =
↪bswap_64(UNIT_PTR(s)[1]);
278     #endif
279 }
280
281 mh_decl void bswap128_block(void *d, const void* s)
282 {
283     #if UNIT_BITS == 8
284         uint_8t t;

```

```

285     t = UNIT_PTR(s)[0]; UNIT_PTR(d)[0] = UNIT_PTR(s)[15]; UNIT_PTR(d)
↪[15] = t;
286     t = UNIT_PTR(s)[1]; UNIT_PTR(d)[1] = UNIT_PTR(s)[14]; UNIT_PTR(d)
↪[14] = t;
287     t = UNIT_PTR(s)[2]; UNIT_PTR(d)[2] = UNIT_PTR(s)[13]; UNIT_PTR(d)
↪[13] = t;
288     t = UNIT_PTR(s)[3]; UNIT_PTR(d)[3] = UNIT_PTR(s)[12]; UNIT_PTR(d)
↪[12] = t;
289     t = UNIT_PTR(s)[4]; UNIT_PTR(d)[4] = UNIT_PTR(s)[11]; UNIT_PTR(d)
↪[11] = t;
290     t = UNIT_PTR(s)[5]; UNIT_PTR(d)[5] = UNIT_PTR(s)[10]; UNIT_PTR(d)
↪[10] = t;
291     t = UNIT_PTR(s)[6]; UNIT_PTR(d)[6] = UNIT_PTR(s)[ 9]; UNIT_PTR(d)
↪[ 9] = t;
292     t = UNIT_PTR(s)[7]; UNIT_PTR(d)[7] = UNIT_PTR(s)[ 8]; UNIT_PTR(d)
↪[ 8] = t;
293     #elif UNIT_BITS == 32
294         uint_32t t;
295         t = bswap_32(UNIT_PTR(s)[0]); UNIT_PTR(d)[0] = bswap_32(UNIT_PTR(
↪s)[3]); UNIT_PTR(d)[3] = t;
296         t = bswap_32(UNIT_PTR(s)[1]); UNIT_PTR(d)[1] = bswap_32(UNIT_PTR(
↪s)[2]); UNIT_PTR(d)[2] = t;
297     #else
298         uint_64t t;
299         t = bswap_64(UNIT_PTR(s)[0]); UNIT_PTR(d)[0] = bswap_64(UNIT_PTR(
↪s)[1]); UNIT_PTR(d)[1] = t;
300     #endif
301 }
302
303 /* platform byte order to big or little endian order for 16, 32 and
↪64 bit variables */
304
305 #if PLATFORM_BYTE_ORDER == IS_BIG_ENDIAN
306
307 #   define uint_16t_to_le(x) (x) = bswap_16((x))
308 #   define uint_32t_to_le(x) (x) = bswap_32((x))
309 #   define uint_64t_to_le(x) (x) = bswap_64((x))
310 #   define uint_16t_to_be(x)
311 #   define uint_32t_to_be(x)
312 #   define uint_64t_to_be(x)
313
314 #else
315
316 #   define uint_16t_to_le(x)
317 #   define uint_32t_to_le(x)
318 #   define uint_64t_to_le(x)
319 #   define uint_16t_to_be(x) (x) = bswap_16((x))
320 #   define uint_32t_to_be(x) (x) = bswap_32((x))
321 #   define uint_64t_to_be(x) (x) = bswap_64((x))

```



```
322
323 #endif
324
325 #if defined(__cplusplus)
326 }
327 #endif
328
329 #endif
```