ETSI TR 103 642 V1.1.1 (2018-10)

CYBER;
Security techniques for protecting software in a white box model

Reference
DTR/CYBER-0029

Keywords
security, software

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

When one tries to protect some sensitive data or code in a device like a mobile, one has to consider that the environment may be compromised (or if it is not yet, it will be soon). The attacker, trying to get secrets of an application, is able to read a lot about the software code and applications, and overrule the security measures, etc. This situation makes much harder the protection of the assets of a mobile application.

To address this, the main strategy is to apply obfuscation and anti-tampering techniques on the software (to hide code, data, key, etc.) and to deploy on the server side assurance function to detect fraudulent behaviour of the application and/or the device. One needs to put in place functions to diversify the software, the protection techniques, the assets (per user or per instance of the application) in order to prevent large scale attacks and functions for quick redeployment, replenishment of the software.

This methodology relies on two pillars:

- White Box Cryptography (which is about protecting the cryptographic functions in the code, together with the key); and
- Code and data protection (which is about making sure that the code, used to run the application cannot be understood, tampered, extracted, exploited, and the data cannot be retrieved and/or modified).

By combining those countermeasures, (and having a correct security design for the application), the level of resistance of a software application to software attack is increased.

1 Scope

The present document reports on the application of techniques for protecting software implementations, in the form of applications and content, using software resident security techniques. The present document makes recommendations for the application of specific techniques including white box cryptography (WBC), code obfuscation, and other techniques denoted as anti-xxx and including anti-tampering, anti-reversing, anti-debugging, anti-cloning, etc. These techniques address the threats presented by attackers of the forms outlined in the present document.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] C. Collberg, C. Thomborson and D. Low: "A taxonomy of obfuscating transformations", Technical Report #148, University of Auckland, 1997.

NOTE: Available at http://profs.sci.univr.it/~giaco/download/Watermarking-Obfuscation/Obfuscation%20Taxonomy.pdf.

[i.2] N. Eyrolles, PhD: "Obfuscation with Mixed Boolean-Arithmetic Expressions: Reconstruction, Analysis and Simplification Tools", Cryptography and Security, Université Paris-Saclay, 2017.

NOTE: Available at https://tel.archives-ouvertes.fr/tel-01623849/.

[i.3] C. Collberg, C. Thomborson and D. Low: "Manufacturing cheap, resilient, and stealthy opaque constructs", University of Auckland, 1998.

NOTE: Available at https://www2.cs.arizona.edu/~collberg/content/research/papers/collberg98manufacturing-clean.pdf.

[i.4] E. Biham and A. Shamir: "Differential cryptanalysis of DES-like cryptosystems", A. Menezes and S. A. Vanstone, Eds., CRYPTO, LNCS 537, pp. 2-21. Springer, 1990.

[i.5] A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel: "A toolbox for cryptanalysis: Linear and affine equivalence algorithms", in E. Biham, Ed., EUROCRYPT, LNCS 2656, pp. 33-50. Springer, 2003.

[i.6] O. Billet, H. Gilbert, and C. Ech-Chatbi: "Cryptanalysis of a White Box Implementation", in H. Handschuh and A. Hasan, Eds., Selected Areas in Cryptography 2004, LNCS 3357, pp. 227-240. Springer, 2005.

[i.7] L. Tolhuizen: "Improved cryptanalysis of an AES implementation", Proceedings of Symposium on Information Theory in the Benelux 2012.

[i.8] T. Lepoint, M. Rivain, Y. De Mulder, P. Roelse, and B. Preneel: "Two attacks on a white-box AES implementation", in T. Lange, K. Lauter and P. Lisonek, Eds., Selected Areas in Cryptography 2013, LNCS 8282, pp. 265-285, Springer, 2014.

[i.9] W. Michiels, P. Gorissen, H.D.L. Hollmann: "Cryptanalysis of a generic class of white-box implementations", Proceedings of Selected Areas in Cryptography (SAC) 2008, LNCS 5381, pp. 414-428, 2009.

[i.10] Y. De Mulder, B. Wyseur, and B. Preneel: "Cryptanalysis of a perturbated white-box AES implementation", in G. Gong and K. C. Gupta, Eds., INDOCRYPT 2010, LNCS 6498, pp. 292-310. Springer, 2010.

[i.11] Y. De Mulder, P. Roelse, and B. Preneel: "Cryptanalysis of the Xiao-Lai white-box AES implementation", in L. R. Knudsen and H. Wu, Eds., Selected Areas in Cryptography 2012, LNCS 7707, pp. 34-49. Springer, 2012.

[i.12] P. Kocher, J Jaffe, and B. Jun: "Differential Power Analysis", Advances in Cryptology - Crypto '99 Proceedings, LNCS 1666, M. Wiener, Ed., pp. 388-397, Springer 1999.

[i.13] E. Biham and A. Shamir: "Differential fault analysis of secret key cryptosystems", Advances in Cryptology - Crypto '97 Proceedings, LNCS 1294, B. Kaliski, Ed., pp. 513-525, Springer 1997.

[i.14] J. Bos, C. Hubain, W. Michiels, and P. Teuwen: "Differential Computation Analysis: Hiding Your White-Box Design is Not Enough", Cryptology ePrint Archive, Report 2015/753, 2015.

NOTE: Available at https://eprint.iacr.org/2015/753.

[i.15] M. Jacob, D. Boneh, E.W. Felten: "Attacking an obfuscated cipher by injecting faults. Proceedings of security and privacy in Digital Rights Management (DRM)", LNCS 2696, pp. 16-31, Springer, 2002.

[i.16] CHES 2017: "Capture the Flag Challenge", organised by the ECRYPT-CSA project, Submission server developed by CryptoExperts, Submission server, hosted by TU Eindhoven.

NOTE: Available at https://ches.2017.rump.cr.yp.to/a905c99d1845f2cf373aad564ac7b5e4.pdf.

[i.17] E. Diehl: "Securing Digital Video - Techniques for DRM and Content Protection", Springer, 2012. ISBN 978-3-642-17344-8.

[i.18] Motion Picture Laboratories Inc.: "MovieLabs Specification for Enhanced Content Protection", Version 1.1, 2015.

[i.19] H. Benoit: "Digital Television - Satellite, Cable, Terrestrial, IPTV, Mobile TV in the DVB Framework", 3rd edition. Elsevier, 2008. ISBN 978-0-240-52081-0.

[i.20] ETSI TS 100 289 (V1.1.1): "Digital Video Broadcasting (DVB); Support for use of the DVB Scrambling Algorithm version 3 within digital broadcasting systems".

[i.21] CI Plus LLP, CI Plus Specification: "Content Security Extensions to the Common Interface", v1.4.2, 2016.

[i.22] EMVCo: "EMV® Mobile Payment - Software-based Mobile Payment Security Requirements", Version 1.0, December 2016.

NOTE: Available at https://www.emvco.com/wp-content/EMVCo-Software-based-Mobile-Payment-Security-Requirements_V1.0_20161213.pdf.

[i.23] ISO SC27: "Study Period on Security requirements, test and evaluation methods for White Box Cryptography (WBC)" October 2016.

[i.24] ISO SC27: "Study Period on Security properties, test and evaluation guidance for White Box Cryptography (WBC)", April 2018.

[i.25] FIDO Alliance, FIDO Authenticator certification program.

NOTE: Available at https://fidoalliance.org/certification/authenticator-certification-levels/.

[i.26] OWASP, Mobile TOP 10 attacks, 2016.

NOTE: Available at https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.

[i.27] Visa Mobile publications, Cloud-Based Payments Program.

NOTE: Available at https://technologypartner.visa.com/Mobile/MobilePublications.aspx#59.

3 Definitions of terms and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

NOTE: The definitions below are split in two paragraphs, to distinguish software protection techniques from attacks.

anti-cloning: software protection technique to prevent execution of a binary on a non-genuine device

NOTE: Strictly speaking the binary knows fingerprints of the execution environment like the MAC address, CPU ID, HDD serial number, etc. and verifies its integrity during execution. If the fingerprints do not match, the device is seen as "cloned" and the program can react in several ways. This technology can help to prevent code lifting.

anti-debugging: software protection technique to prevent debugging of an application through a debugger

NOTE: Example of debuggers are GDB, WinDBG. If a debugger is detected the program can react in several ways like notification of the user or abortion of the execution.

anti-disassembly: method to prevent disassembling of the binary through disassemblers

NOTE: Disassemblers like IDA Pro, Binary Ninja, etc. Anti-Disassembly can be applied directly on assembler code to let disassemblers generate wrong opcodes or on control flow level to hide the connection of the basic blocks of a function.

BGE: named after its authors (Billet, Gilbert and Ech-Chatbi), a realistic (in terms of work factor) algebraic attack on the white-box AES design by Chow, Eisen, Johnson and van Oorschot

NOTE: It has been further generalized to all Substitution-Permutation Network ciphers.

cloning: making an illegal copy of the contents (of the device) to a new device or for analysis

code anti-tampering: software protection technique that prevents modification of a binary

NOTE 1: If a modification of the binary is detected, the program can react in several ways like notification of the user or abortion of the execution.

NOTE 2: Code anti-Tampering can be divided into two groups:

- Static Anti Tampering: Before execution of the binary the integrity of the file will be verified and loaded into memory if and only if the checksum is valid.
- Dynamic Anti Tampering: During execution of the binary random integrity checks will be executed to self-check if the binary was tampered.

code/data anti-reversing: set of software techniques aiming to protect an attacker having access to a program to understand its implementation or having access to sensitive data in clear form

NOTE: See also Obfuscation.

code-flow/-pointer integrity: technique which prevents attacks that try to take control of the execution flow of the program

NOTE: During compilation the compiler injects guards that will verify if the destination address is valid. If the destination address is not valid the program can detect that and abort the execution.

code lifting: method where the attacker tries to reuse part of the binary code for executing it in an unauthorized way

code obfuscation: combination of several software based techniques that transform the source or machine code into code that is very difficult to understand for humans

NOTE: The purpose of obfuscation is to harden the resistance against attacks like reverse engineering or tampering of the code. Obfuscation can be broken but the amount of time and the expertise to apply attacks on obfuscation can be significant and is done by skilled experts.

code-pointer separation: technology to verify the integrity of code pointers by splitting the memory into safe memory and regular memory.

NOTE: Code pointers are placed in the safe memory and the integrity of the memory will be verified during execution of the binary. This technique prevents attacks where an attacker tries to take control of the execution flow of the application and has a performance advantage over Code-flow integrity.

code watermarking: technique to deeply embed a unique watermark in a binary and that is very difficult to detect and remove

NOTE: The watermark can be used to authenticate a genuine software but also to trace back illegal copies.

data anti-tampering: software protection technique that prevents modification of data (e.g. integrity checksum, redundancy, etc.)

NOTE: Data anti-tampering can be static or dynamic.

data encodings: reversible method to transform data without the requirement of a secret key

NOTE 1: Typically used in WBC.

NOTE 2: They are external encodings applied at the input and at the output of a crypto functions. They are internal encodings applied to internal variables and/or tables. The purpose is to manipulate data not in clear form and to avoid software side channel attacks. They are static encodings that are fixed at compilation time and they are dynamic encodings that are changing at execution time.

data lifting: method where the attacker tries to reuse data in an unauthorized way, on behalf of the genuine data owner

NOTE: Typically re-using a WBC dynamic key from one instance to another.

data obfuscation: set of software protection techniques to hide data against an attacker trying to reverse it (i.e. find, detect, and/or localise)

NOTE: This includes hiding sensitive data (encryption/encoding), making, splitting input data into smaller chunks or using steganography.

data protection: techniques that try to hide data like cryptographic keys, debug strings, application strings, etc. against an attacker and tools like "strings" or an (hex-) editor

data watermarking: technique to deeply embed a unique watermark in a data that is very difficult to detect and remove

NOTE: An example is a movie file watermarked to track back illegal copies.

device binding: software and/or a hardware technique to bind a data or code to a device, meaning if one tries to use the bound item in another device, it will fail

Differential Computation Analysis (DCA): side-channel attack derived from DPA aiming at key extraction from a WBC where measuring noisy power consumption is replaced by noiseless intermediate computation value collection during a cryptographic software executions

Differential Fault Analysis (DFA): family of attacks that relies on the analysis of the difference in outputs when a perturbation is injected during a cryptographic computation

NOTE: Such a perturbation can be a laser shot on a hardware chip, or a value modification in the context of a software implementation.

Differential Power Analysis (DPA): attack that exploits the information leakage of the manipulation of a secret value by a hardware, through its power consumption

NOTE 1: Information is collected for multiple variable inputs with an oscilloscope. Difference of means is then computed, after splitting the dataset based on a guessed value computed from the actual inputs and a key hypothesis. The good key stands out as it correctly splits the data.

NOTE 2: Power consumption is not the only side-channel that can be used. The same attack based on electro-magnetic radiation would be called DEMA, for example.

hooking: range of techniques used to alter or augment the behaviour of an operating system, of applications, or of other software components by intercepting function calls or messages or events passed between software components

stack cookies: method used to protect against buffer overflows

NOTE: The compiler integrates known values, "stack cookies", between a buffer and the control data. When the stack cookie gets overwritten during a buffer overflow the code can detect that by verifying the stack cookie with the hardcoded value in the code and abort the execution of the program.

White Box Cryptography (WBC): set of software protection techniques aiming at protecting software implementations of cryptographic algorithms against key recovery

NOTE: The specificity of a white-box cryptography attacker is that he is assumed to have full control over the whole execution of an implementation. In this highly unfavourable setup, security by obscurity was the initial industry response, with the deployment of private algorithms in real-world WBC. The academic world effort has increased over the past years, with a focus on standard cryptographic algorithms.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AES Advanced Encryption Standard
Anti-xxx All methods starting with "anti-"

NOTE: It includes at least anti-tampering, anti-cloning, anti-reversing, and anti-debugging.

BGE Billet, Gilbert and Ech-Chatbi

NOTE: From the name of the authors of a well-known attack against a white-box implementation of AES.

CA Conditional Access
CA/DRM Conditional Access/Digital Rights Management
CBP Cloud Based Payment
CHES Cryptographic Hardware and Embedded Systems

NOTE: A conference on cryptographic hardware and embedded systems.

CPE Customer Premises Equipment
CPUID derived from CPU (Central Processing Unit) IDentification
DCA Differential Computation Analysis
DES Data Encryption Standard
DFA Differential Fault Analysis
DPA Differential Power Analysis
DRM Digital Rights Management
DVB Digital Video Broadcasting
ECI Embedded Common Interface
EMV Europay Mastercard Visa
IC Integrated Card
ISG Industry Specification Group
MAC Medium Access Control
MBA Mixed Boolean-Arithmetic
OS Operating System
OTT Over-The-Top
SE Secure Element

SIM	Subscriber Identity Module
SoC	System on Chip
VCK	Virtual Car Key
WBC	White Box Cryptography

4 Threat security model

There are different threat security models used in the field of software security.

The classical threat model involves a malicious third party Eve attempting to decrypt the communication between Alice and Bob. In this case, Alice and Bob are friendly parties communicating, and not attacking the system (Eve is an external attacker). There are situations where this is not true, and where the attacker may be one of the two communicating parties.

In a "Black-box model" (see figure 1), the attacker is assumed to have access to inputs and outputs, to be able to observe and modify the communication (read, intercept and/or substitute). He has no access to the encryption keys, or more generally to the system performing cryptographic operations. He has no access to the software binary or to the execution environment internals, and may abuse the software functionality.
The implementation stays opaque ("black").

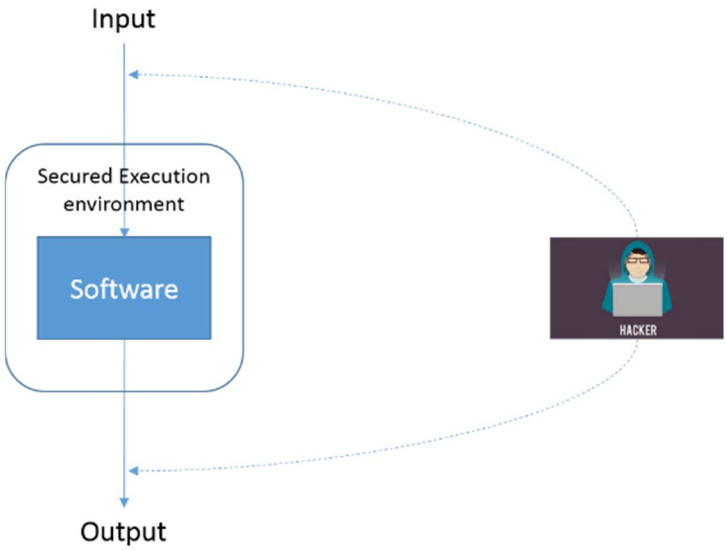

Figure 1: Black-box threat model

In a "Grey-box threat model" (see figure 2), the attacker has the same power as in the black-box model, but he also partially knows and/or have access to the internal structure of the system performing cryptographic operations (e.g. access to the documentation of the internal data structure, the cryptographic algorithms used). He has an indirect access, through side channel analysis, to execution environment internals and may disrupt the software execution through faults. In this model, the attacker does not have access to the keys and/or cannot tamper with the cryptographic algorithms.

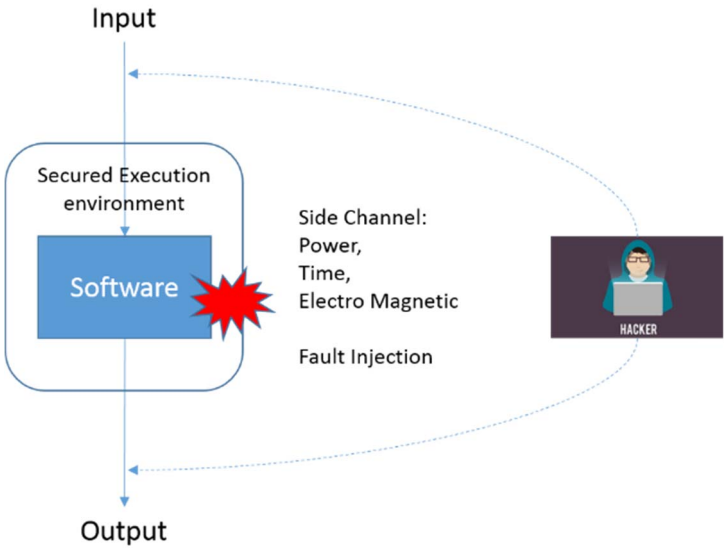

Figure 2: Grey-box threat model

In a "White-box model" (see figure 3), the attacker has the same power as in the grey-box model, and more: he has full knowledge of the internal structure of the system performing cryptographic operations, and is allowed to alter/tamper it. He has access to the software binary, a direct access to the execution environment internals, he can make trace, reversing, hooking, etc.

The implementation is clear ("white").

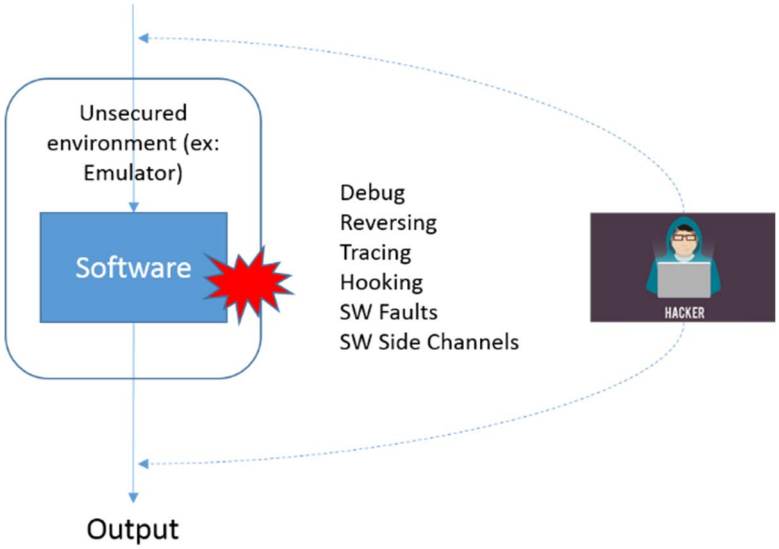

Figure 3: White-box threat model

5 Description of the different techniques

5.1 Introduction

There are several methods to protect software application, code and data. These software methods are usually split in 2 main branches:

- White Box Cryptography; and

- Code and data protection.

5.2 White Box Cryptography (WBC)

Originally, this technology was used for the Digital Right Management use case, but it became more popular in 2015 when some Payments Schemes chose to deploy some mobile banking applications based on software only. The WBC was born in 2002, and its objective is to hide a key, and make sure that no one can extract it from the code. As of today, the security of WBC relies on the fact that it is based on secret design. Because it is software based, and because some constant researches are unveiling new attacks, the WBC is also a moving science.

WBC can work with both static keys (embedded in the code) and encrypted dynamic keys (loaded and decrypted at run time). Figures 4 and 5 depict a high-level overview of WBC, for the case of a static and dynamic key implementations.

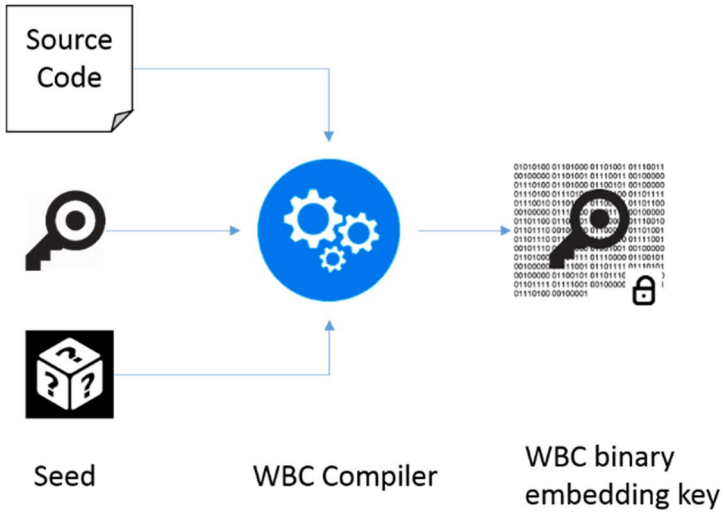

Figure 4: WBC with static key

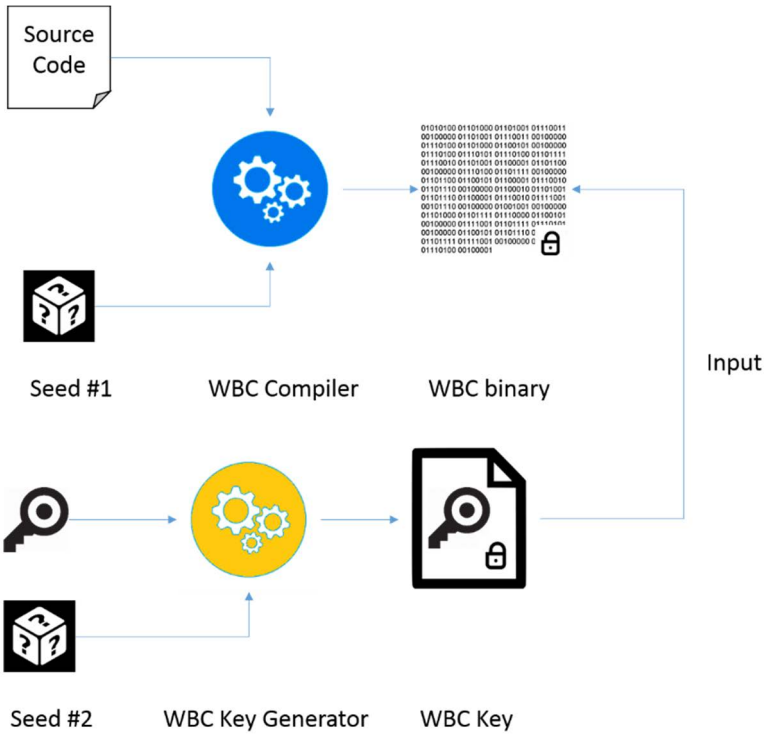

Figure 5: WBC with dynamic key

WBC is one of the building block for software security strategy. Since its publication in 2002, with focus only on block cipher algorithms, it has gained momentum, and has evolved to cover more cryptographic algorithms.
Evaluation methodology is under construction in ISO ([i.23] and [i.24]), certification process is on-going in FIDO ([i.25]), showing a great interest in this technique, not only from academics but also from industry.

5.3 Code and data protection

5.3.1 Introduction

In order to re-inforce WBC implementations, some smart techniques are deployed.

As software protections are implemented as code injected into the original software, one needs to protect the protections themselves to slow down an attacker. For example, one can protect code with anti-debug and apply on top of it anti-tamper, to slow down an attacker from tampering the anti-debug protection. Also, obfuscation makes protections more difficult to understand.

The combination of such different techniques makes a global secure solution, as highlighted in figure 6.

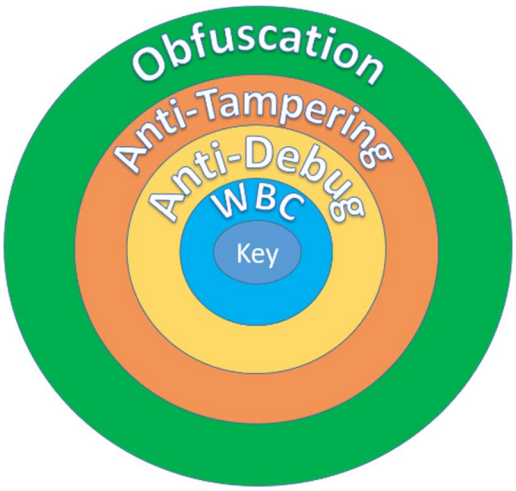

Figure 6: Layered protections

5.3.2 Anti-xxx techniques

Some of these techniques are also known as anti-xxx. To make sure that the WBC code itself is not tampered, a variety of measures are deployed:

- anti-reversing, to guarantee that the code cannot be reversed engineered,
- anti-tampering to keep code constituency and integrity,
- anti-debug, to avoid debugging mode leaking information, and
- last but not easiest anti-cloning, to ensure that the code is executed on a genuine environment.

5.3.3 Code and data obfuscation

For a complete taxonomy of obfuscating transformations, see annex A or [i.1].

Code obfuscation is a technique to slow down, or ideally completely hinder, a potential attacker/reverse engineer from understanding a piece of software, by applying semantics-preserving transformations on the given code.

These transformations include:

- Complicate code:

  - By substituting operations with longer/more nested ones (e.g. MBA formulae [i.2]) or make use of indirect control transfers.

- Re-arrange code:
  - By shuffling independent code parts for diversification.
- Re-structure code:
  - By splitting existing connections within the code.
- Duplicate code:
  - By cloning existing code parts and re-using them.
- Introduce bogus control flow:
  - by connecting random code paths use so called opaque predicates [i.3].
- Re-naming/stripping of symbolic names/variable names, etc.
- Breaking/hindering analysis tools:
  - By stressing the limits or weaknesses of popular tools an attacker may use (i.e. in the simplest case anti-disassembly techniques).

While the combination of all these transformations can significantly complicate code analysis, most of them also induce a size and/or runtime overhead in the resulting binary. So ultimately the level of security one can achieve is bounded by the allowed resources the resulting binary may consume.

Data obfuscation is a technique to slow down, or ideally completely hinder, a potential attacker/reverse engineer from finding or understanding data inside an application. The attacker could use the data to either understand the control flow of the application or to rebuild specific parts of the code. The data an attacker is interested in could be cryptographic keys, debug strings, application strings, etc.

These transformations can be used to protect data:

- Encryption:
  - The data in the application is stored encrypted. At first use the application decrypts the data and use it. If possible the data is encrypted again after use.
- Homomorphic operations:
  - Homomorphic encryption can be used to apply operations on the data without leaking the clear data. For that the data is stored encrypted in the application. During execution the application uses homomorphic operations on the data so the data never appears clear.
- Substitution:
  - The data is substituted with a given dictionary and stored inside the application. At first use the data is substituted again and used by the application. This method has the advantage that it is much faster on execution than using encryption.
- Embedding data in code:
  - The data is split up into machine word constants and stored in operations in the control flow. During execution of the code the data is rebuilt from the constants in the control flow. The method has the benefit that code obfuscation can be applied afterwards to even make the protection stronger.
- White box:
  - See white box cryptography (clause 5.2).

5.3.4 Device binding

Anti-cloning is the set of techniques to prevent data and code cloning. Device binding is one of the methods for implementing anti-cloning.

In order to prevent the attacker from executing the application/library on a different device than the device the application/library was deployed to and steal the identity or data from the device, it is important to protect against this kind of attacks. Think about keys inside a WBC that cannot be recovered. The attacker could try to copy the code of the WBC into his library (code lifting) and use the code to encrypt/decrypt the messages without even needing to know the algorithm and/or the key. Another case would be that a paid application could be copied and stolen to run on other devices. To prevent this kind of attack the following techniques can be applied:

- Device binding:
  - Find constant device specific characteristic (CPUID, Ethernet MAC, Disk Drive Serial Number) and use them to verify the environment.
- Device specific code obfuscation:
  - A more advanced technique is to generate the application with the knowledge about the environment to make use of the characteristic in the code flow. This makes it much harder for the attacker to identify the spots for the checks.
- Device specific license generation:
  - Each device retrieves the same application but a license is necessary to run the application. The license is generated by the authentication server on the first run with knowledge about the environment of the application. The generated license does not only contain a key that allows the application to run on the device but also contains device specific constants/code parts that are only valid on the device the license was generated for.

5.3.5 Watermarking

In order to prevent the leakage of an application, watermarking is a technology that can be very useful. The application is marked with different hidden tags/checks that are hard to identify by the attacker. Watermarking can also raise the security of an application. The fact that the application contains watermarking can be seen as a psychological factor for the attacker to not attack or release the application/code at all, as it can disclose a track back to the attacker.

Some common techniques are:

- Code diversification for each customer:
  - By generating a build of an application for each customer the binary looks different all the time. This fact cannot be easily removed as it would need binary recompilation to generate a new layout of the binary and code to remove the watermarking.
- Hidden Constants:
  - Constants used in calculations can be encoded/marked with customer specific information. The constants are needed to calculate the right result and so cannot be easily removed.

5.3.6 Software-based exploit mitigation

Since bugs are prevalent in today's software and also will be in future software, due to the human nature and the design of the programming languages used, exploit mitigation techniques can help mitigate, and in some instances completely render, code execution or information leaking attacks. In contrary to previously mentioned techniques, which only protect software and its data from an attacker, exploit mitigation techniques also protect the environment in which this software is run and thereby in return protect other software run on the same device. The process of exploitation aims at abusing and leveraging found bugs in software to either execute different code or read or write unauthorized from and to data. While other exploit mitigation techniques, such as data execution prevention or address space layout randomization depend on the given hardware and operating system the following techniques can be deployed and integrated on a per-binary basis, independent of external factors and aim at raising the complexity for such attacks.

- Stack cookies:
  - Detect and prevent stack-based buffer overflows.
- Code-flow/-Pointer integrity & Code-Pointer separation:
  - Detect and prevent pointer-related exploitation.
- Software diversity:
  - Make use of a light form of obfuscation, thereby making code-reuse attacks (such as return-oriented programming) impossible, without prior information leaks.

While the first two are already supported by the most popular compilers and can be integrated seemingly into new software, only the latter one, software diversification, is a relatively new concept and not yet supported by any compiler, which also allows new as well as already present and compiled binaries to be protected.

6 Description of attacks

6.1 Introduction

The strategy of the attacker consists in finding the weakest or easiest path.

The techniques described in the previous clause should be combined to obtain an overall good protection.

6.2 General description of attacks on a software application

One of the main threat for a software application, such as banking, payment, or authentication, is running on a compromised OS, able to monitor and manipulate the application memory usage. This is why the main hypothesis for software protection is self-protection, not relying on the OS security features.

Table 1 is largely inspired from [i.22] which describes attacks on mobile application, but can be extended to software applications, together with the countermeasures, as described in clause 5.

Table 1: Attacks and mitigation

Attacks	Description	Attack Path	Mitigation	Clause reference
Reverse engineer software application	Extract sensitive software Application assets	Reverse engineer the binary code to understand its functionality. This can be done statically through a disassembler and/or a decompiler or dynamically with tracing or with an emulator	Anti-reverse using all software protection techniques described	5.3
Monitor assets, control flow and API	Recover sensitive data like keys or strings	An attacker can create a proxy library that will monitor the public API of the shared library. Another way is to directly hook the APIs or break at a specific location to retrieve data	WBC Data obfuscation Anti-tampering Anti-debug Anti-hooking	5.3
Modify the software application code statically or dynamically	Alter behaviour of Software Application	Code is modified or malicious code injected, for example to present false information to the user or let the application behave in the desired way	Anti-tampering Obfuscation Anti-hooking Anti-Emulation Anti-reversing Anti-disassembly Watermarking	5.3
Exploit interfaces between components	Abuse the interfaces between the components that make up the application	Masquerade as a legitimate application in the REE interacting with a trusted application in the TEE (if any) Compromise the results of user authentication	Secure channel between genuine caller/callee components, protection of assets with WBC, protection of code with Anti-xxx Authentication Authorization	
Extract assets during runtime	Recover assets from an application running under the attacker control	The application is executed under debugger, emulator or dynamic binary instrumentation (DBI). The attacker intercepts plaintext assets at the time they are processed in memory	WBC (Key) Obfuscation Anti-debug Anti-emulation	5.2 5.3
Cloning of data to a different device	Clone the data of one device to another to pretend to be the same device	The attacker copies the application and the user data from one device to another to act as the cloned device	Device binding Anti-emulation Watermarking	5.3
Illegal or unauthorized application copy	Copy the application from the legal device to another one	The attacker copies the application to another device to use it without having the right to do it	Watermarking Device binding Anti-emulation	5.3

Another interesting reference is OWASP Top 10 attacks for a mobile application [i.26]. It includes, for 2016:

- improper usage platform (M1),
- insecure data storage (M2),
- insecure communication (M3),
- insecure authentication (M4),
- insufficient cryptography (M5),
- insecure authorization (M6),
- client code quality (M7),
- code tampering (M8),
- reverse engineering (M9), and
- extraneous functionality (M10).

This attack classification is more or less covered by the previous.

6.3 Focus on attacks on white box implementations

6.3.1 Cryptanalysis

Cryptanalysis assumes that the attacker understands the design of the white-box implementation. For example, the attacker can have access to the specification of the white-box implementation or the attacker can have reverse engineered the implementation. Cryptanalysis typically uses methods that were developed to analyse the security of symmetric ciphers in a black-box model. Examples of such methods are differential cryptanalysis [i.4] or algorithms that can compute linear and affine equivalences [i.5]. This has resulted in very efficient attacks on white-box implementations.

The first cryptanalysis of a white-box implementation was presented in [i.6] and is widely-known as the BGE attack. The BGE attack can be used to extract an AES key from the implementation of Chow et al. with a work factor of 2^{30}. Based on the work presented in [i.7], this work factor was later improved to 2^{22} in [i.8]. The BGE attack can also be applied to extract the key if the implementation uses secret external encodings and it allows the attacker to determine the values of these encodings with little additional effort. The BGE attack was later generalized to an attack on a more general class of symmetric algorithms, also referred to as substitution-linear transformation ciphers [i.9].

After the BGE attack several other white-box implementations of AES that were claimed to be resistant against this attack have been published. However, apart from very recent proposals of which the security still needs to be analysed, all these implementations have been broken [i.7], [i.10] and [i.11]. These references present new efficient attacks and show that one of these implementations is still vulnerable to the BGE attack. They report work factors of 2^{22}, 2^{16} and 2^{32} respectively.

6.3.2 From Side-channel analysis to Differential Computation Analysis

The second type of attack is based on side-channel analysis. These techniques were first developed for analysing the security of an implementation of a cryptographic algorithm in the presence of a side-channel. Examples of side-channels are computation time and power consumption. The corresponding attack model is commonly referred to as the grey-box model. Smart card ICs are a typical example of an environment to which a grey-box model can apply. The main prominent example of side-channel analysis is Differential Power Analysis (DPA) [i.12].

Since the white-box model is stronger than the grey-box model, side-channel analysis can be applied directly to white-box implementations. However, in the white-box model the attacker has generally more and more accurate information and more capabilities. This is exploited in Differential Computation Analysis (DCA) which was introduced in [i.14]. DCA is based on DPA; however, instead of examining power usage, DCA uses computation traces. For example, such traces can comprise contents of memory or registers.

An important difference with cryptanalysis is that side-channel analysis does not require any knowledge about, or understanding of, the implementation. This means that the attacker does not need access to the specification of the white-box implementation and that no reverse engineering is required to mount a side-channel analysis attack. The attacks, at least in their basic form, can also be easily automated. After this, little knowledge or skill is required to apply these attacks. Side-channel analysis can usually be made more effective if the attacker has more understanding about the implementation.

To the current state of knowledge, the use of external encodings is generally a good measure against side-channel analysis of white-box implementations.

6.3.3 Fault Analysis

The third type of attack is based on fault analysis where an attacker may also be able to introduce faults during the execution of the algorithm and analyse the results. The main prominent example of fault analysis is Differential Fault Analysis [i.13]. The first DFA attack on a white-box implementation was presented in [i.15].

In the white-box model, the attacker has now more control over the faults that are introduced during the computations. Indeed, any type of modification can be applied during the execution of the crypto algorithm which means that all possible fault models are relevant. Beyond classical DFA, the attacker can use fault analysis to deactivate white-box cryptography protections, e.g. protections against DCA. Then this leads to combined attacks that become easily practical compared to the usual context of hardware-protected implementations. Also, fault analysis techniques can be used by the attacker to reverse the code and get knowledge about the design of the white-box cryptography implementation. For this type of attacks, it is often required for the attacker to reverse (at least partially) the white-box implementation in order to perform exploitable fault attacks. This type of attack is more difficult to automatize compared to DCA.

6.3.4 CHES 2017 challenge

In 2017 the ECRYPT-CSA consortium organized a public white-box contest [i.16]. The contest invited designers to submit the C code of their white-box AES implementation, and attackers were invited to cryptanalyse these implementations; in other words, to extract the AES key from each of the submitted implementations.

The evaluation conditions in the challenge are not the same as for products in the field:

- other software protections are used in conjunction to WBC,
- and the focus here is only on AES, with specialized tools while WBC can also be applied to other types of algorithms.

The results of the contest were presented during the rump session of the CHES 2017 conference [i.16]. In total there were around 100 submissions. All these implementations were broken: one implementation took 28 days to break and all other implementations were broken in 12 days or less. However, the use of external encodings was not allowed, which means that this was the most challenging case from the designer's point of view. In addition, more research is required to analyse the quality of the submitted implementations since designers and attackers did not need to disclose any details about the implementations and the attacks.

7 Use cases

7.1 Digital Rights Management

A Digital Rights Management (DRM) system can be used to control access to digital content. To achieve this, the DRM system encrypts the content before distributing it to the Customer Premises Equipment (CPE), and a DRM client in the CPE can only decrypt the content if a corresponding license has been received and if all conditions in the license are satisfied. The reader is referred to [i.17] for detailed information about DRM systems and content protection.

The attacker has access to the CPE in a DRM system. Furthermore, secure hardware may not be available for implementing the DRM client, e.g. if the CPE is a mobile device or a PC. In such cases the attacker has white-box access to the implementation of the client. Software protection techniques are therefore commonly used to make a DRM client read-proof and tamper-resistant. The use of such techniques can also be required by applicable robustness rules.

In 2015, a company working on film and television technology published a set of requirements for enhanced content protection [i.18]. Compliance with these requirements is generally considered to be important for content protection systems since a member company may require that compliant devices are used to process certain types of their content (such as Ultra HD content). Software protection techniques may be used to satisfy several of the specifications requirements [i.18] (e.g. the software diversity requirement).

The system used to protect Pay-TV content is similar to a DRM system, and is usually referred to as a Conditional Access (CA) system. For detailed information about Pay-TV systems and the widely-adopted DVB Framework, refer to [i.19] or visit www.dvb.org. If the CA system is compliant with the DVB standard, then an important difference with DRM systems is that the content decryption algorithm is required to be implemented in hardware (see also ETSI TS 100 289 [i.20]). This implementation is typically integrated in a System on Chip (SoC) of a set-top box or a CI+ module [i.21]. Traditionally, the CA client (which does not include the content decryption algorithm) was implemented in a smart card IC. However, in recent years the number of deployments of smart card-less CA systems has increased. In addition, the distribution of Over-The-Top (OTT) content has increased, and consequently, the number of implementations of DRM clients in set-top boxes has also become larger. In practice, software protection techniques are used to harden implementations of card-less CA clients and DRM clients in set-top boxes.

The ETSI Industry Specification Group (ISG) on Embedded Common Interface (ECI) has standardized an environment for exchangeable CA/DRM solutions. More specifically, they have developed a standardized system architecture for general purpose, software-based, embedded and exchangeable CA/DRM systems. This initiative may lead to a further increase in the use of software protection techniques to secure the implementation of CA/DRM clients.

NOTE: See https://www.etsi.org/standards-search#TB=810 for ECI publications.

7.2 Automotive

Most car manufacturers have launched a Virtual Car Key (VCK) service. VCK is about replacing the physical car key (or key fob or remote control) by an app on a consumer device. The device is most often a smart phone, but will be more and more any kind of wearable device. The advantage of VCK is that the car keys can be shared remotely without the hurdle of handling a physical object. In addition, VCK can be associated with permissions (like speed limitation or geo-fencing) and can have a limited validity period. Thus, VCK is a service to share a car within a family or with friends but it is also a crucial enabler to manage company fleet or to deploy car sharing / car rental services: the whole renting process can be dematerialized.

In this context, the car manufacturers can no longer control the hardware on which the credentials for VCK are deployed. Some devices will feature tamper resistant secure elements (SE), some other no. And even in the case of a device with a SE, it may be hard to deploy some credentials on it. The car manufacturer are consequently forced to use technologies like WBC to secure their credentials on non-tamper resistant hardware.

This use case may be extended to some other segments of the new sharing economy like flat rental, office sharing, etc.

7.3 Cloud-based payment

Digitization of payment cards within mobile phones has been the hype within issuing banks and payment schemes for the last 10 years, but no solution have been really taking off until today.

Firstly, mobile telecom operators have tried to promote their SIM cards as the right tamperproof element to load payment application and payment credentials. But the solution may be hard to deploy for issuing banks because of a difficult business model to be set-up between them and mobile telecom operators, and because of the technical complexity of the provisioning of the payment credentials.

More recently, a handset manufacturer has launched a proprietary wallet where the tamperproof element is embedded in the mobile phone, providing a security similar to payment chip card or multi-application SIM card. Nevertheless, with this solution, the issuing banks are stuck with the handset maker, and therefore the business model is also not so easy for them.

Lastly, Cloud-Based Payment (CBP) specifications (e.g. [i.27]) have been defined and allow issuing banks to develop their own mobile payment application to digitize payment cards on mobile devices. This solution is not available on all the OS devices. This solution has the benefits to completely free the issuing banks from mobile telecom operators and handset makers, but impose to deploy a secure solution relying on software protection only, or at least on device security features available on all mobile devices targeted.

This solution can therefore not rely for the time being on a tamperproof element from the mobile device, and is a compromise between a medium but sufficient security on the mobile phone side, and a classical issuing bank risk management on the payment authorization server side. This security on the mobile device needs to protect the payment credentials defined by the payment schemes and comply with security evaluations (e.g. [i.22]) done by external independent security labs for any live commercial deployments (certification process).

The security techniques for protecting software in a white box model, described in the present document, are relevant technology candidates to provide the necessary software security protection required by payment schemes as described above.

8 Advantages and drawbacks

The aim of software protection is to harden the expert attacker task and prevent him from:

- understanding what is going on (for a human),

- tampering the software and its resources,
- designing any forensic investigation or debug analysis,
- debugging the software,
- cloning the software and its resources,
- extracting the cryptographic key, and
- abusing the software.

The advantages of software protection techniques include the following:

- easier redeployment than with hardware protection,
- easier update than with hardware protection, and can be remote,
- diversification of the protection to harden exploitation of an attack and its scalability,
- resistance to attacks even if the OS is compromised,
- working on multiple platforms,
- preserving the functionality of the original software, and
- independency from any hardware mechanism.

The drawbacks of software protection techniques include the following:

- decrease the software performance, and
- increase the software size.

9 Conclusion

White box cryptography is an important aspect to the strategy of the cryptographic key protection, but it is also necessary to protect the secured application in which the keys are used. To do so robustly and in a performance-efficient manner, several static and dynamic reverse-engineering mechanisms should be employed. Additional protection tools such as obfuscation are critical to comprehensive security.

Annex A:
Classification of the techniques for obfuscation

Figure A.1 extracted from [i.1] shows the classification of the technical protections (a), the quality of the transformations (b), the information transformed by obfuscation (c) and the details by type of transformation (d), (e), (f) and (g).

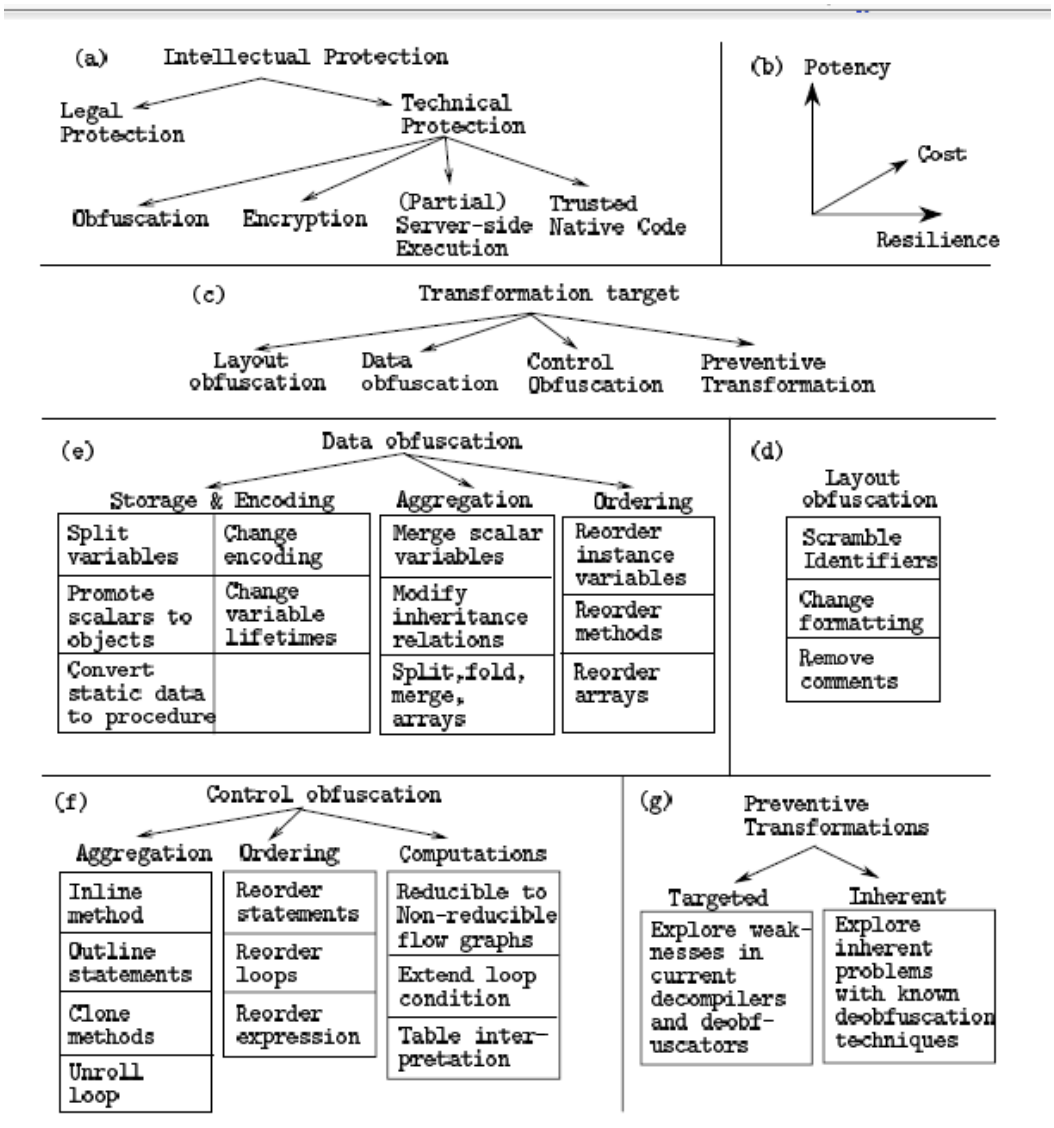

Figure A.1

History

Document history		
V1.1.1	October 2018	Publication