Published in A. Elçi, S.B. Ors, and B. Preneel, Eds, *Security of Information and Networks*, pp. 7–12, Trafford Publishing, 2008.

On White-Box Cryptography

Marc Joye

Thomson R&D France
Technology Group, Corporate Research, Security Laboratory
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné Cedex, France
`marc.joye@thomson.net`

Abstract. White-box cryptography techniques are aimed at protecting software implementations of cryptographic algorithms against key recovery. They are primarily used in DRM-like applications as a cost-effective alternative to token-based protections. This paper discusses the relevance of white-box implementations in such contexts as a series of questions and answers.

Q1: What is white-box cryptography?

A major issue when dealing with security programs is the protection of "sensitive" (secret, confidential or private) data embedded in the code. The usual solution consists in encrypting the data but the legitimate user needs to get access to the decryption key, which also needs to be protected. This is even more challenging in a software-*only* solution, running on a non-trusted host.

White-box cryptography is aimed at protecting secret keys from being disclosed in a *software* implementation. In such a context, it is assumed that the attacker (usually a "legitimate" user or malicious software) may also control the execution environment. This is in contrast with the more traditional security model where the attacker is only given a black-box access (i.e., inputs/outputs) to the cryptographic algorithm under consideration.

Q2: What is the difference with code obfuscation?

Related and complementary techniques for protecting software implementations but with *different* security goals include code obfuscation and software tamper-resistance. *Code obfuscation* is aimed at protecting against the reverse engineering of a (cryptographic) algorithm while *software tamper-resistance* is aimed at protecting against modifications of the code.

All these techniques have however in common that the resulting implementation must remain directly executable.

Q3: How realistic is the white-box threat model?

The traditional (i.e., black-box) threat models for encryption schemes are the chosen-plaintext attack (CPA) model and the chosen-ciphertext attack (CCA) model. In the *CPA model*, the adversary chooses plaintexts and is given the corresponding ciphertexts; in the *CCA model*, the adversary chooses ciphertexts and is given the corresponding plaintexts.

In the white-box threat model, the adversary can get access to the same resources as in the black-box model *plus* full control of the encryption/decryption software. The goal of the adversary is to extract the key. One may wonder why such a scenario makes sense since an adversary controlling the encryption/decryption software can make use of it to encrypt or decrypt arbitrary data without needing to extract the keys. We note that a white-box implementation can be useful as it forces the user to use the software at hand.[1] Furthermore, other security measures can be used concurrently.

If an adversary could recover the decryption key, then the data could be decrypted and used with *any* software on *any* host (cf. BORE attacks — break once, run everywhere). This would allow a global crack with more severe damages.

Q4: What are the applications of white-box cryptography?

The main application of white-box cryptography is the secure distribution of "valuable" content such as in *digital rights management* (DRM) applications. Here the main goal is to prevent the unauthorized use of bulk data processed by software, like music or movies.

More surprisingly, white-box techniques also allow the development of "lightweight" cryptography — note that only the private operation is "light", for example:

Converting a secret-key encryption into a public-key encryption

It is easy (in principle) to construct a *public-key encryption* scheme from a white-box implementation of a secret-key encryption algorithm E_K, say $\mathrm{WB}(E_K)$. Anyone in possession of $\mathrm{WB}(E_K)$ can encrypt messages while only one possessing secret key K is able to decrypt using decryption algorithm ${E_K}^{-1}$.

Note: To be valid, this transformation requires that E_K and ${E_K}^{-1}$ are different. One should also ensure that the release of $\mathrm{WB}(E_K)$ does not contradict the usual security properties, like one-wayness or better, semantic security.

Transforming a MAC into a digital signature

Dual to above, a *(keyed) message authentication code* (MAC_K) scheme can be used to produce a digital signature. Being given secret-key K, one can compute a signature using MAC_K on any message; further, using a public (and certified) white-box implementation of the "verification" algorithm, say $\mathrm{WB}({\mathrm{MAC}_K}^{-1})$, anyone can verify the validity of signatures. Contrary to

[1] We also note that external encodings can be used so that the encryption (respectively, decryption) software requires encoded inputs and produces encoded outputs. As a result, the white-box implementation cannot be used for evaluating the encryption algorithm (respectively, decryption algorithm) in isolation. See Q5.

traditional MAC constructs, the verification does not require the knowledge of secret K.

Note: Again, in order to satisfy the non-repudiation property, one must assume that the operations of computing and verifying a "MAC" are different. Using a cryptographic hash function h, if E_K and ${E_K}^{-1}$ are different, a signature on message m can e.g. be produced as $S = E_K(h(m))$; its correctness can then verified with $\mathrm{WB}({E_K}^{-1})$ by checking whether $\mathrm{WB}({E_K}^{-1})(S)$ is equal to $h(m)$.

Q5: What is the general methodology behind white-box implementations?

The main idea of white-box implementations is to rewrite a *key-instantiated* version so that all information related to the key is "hidden". In other words, for each secret key, a key-customized software is implemented so that the key input is unnecessary.

Most symmetric block-ciphers, including the AES and the DES, are implemented using substitution boxes and linear transformations. Imagine that such a cipher is white-box implemented as a huge lookup table taking on input any plaintext and returning the corresponding ciphertext for a given key. Observe that this white-box implementation has exactly the same security as the same cipher in the black-box context: the adversary learns nothing more than pairs of matching plaintexts/ciphertexts. Typical plaintexts being 64-bit or 128-bit values, such an ideal approach cannot be implemented in practice.

Current white-box implementations apply the above basic idea to smaller components. They represent each component as a series of lookup tables and insert random input- and output bijective encodings to introduce ambiguity, so that the resulting algorithm appears as the composition of a series of lookup tables with randomized values.

To add further protection, external (key-independent) encodings may be used by replacing the encryption function E_K (respectively, decryption function ${E_K}^{-1}$) with the composition $E'_K = G \circ E_K \circ F^{-1}$ (respectively, ${E'_K}^{-1} = F \circ {E_K}^{-1} \circ G^{-1}$). Input encoding function F and output decoding function G^{-1} (respectively, G and F^{-1}) should not be made available on the platform that computes E'_K (respectively, ${E'_K}^{-1}$) so that the white-box implementation cannot be used to compute E_K (respectively, ${E_K}^{-1}$). Although the resulting implementation is not standard, such an approach is reasonable for many DRM applications.

Q6: Are there alternatives to white-box cryptography?

Tamper-resistant tokens (e.g., smart cards) also help in preventing key recovery attacks. Typically, cryptographic keys are stored in the non-volatile memory of the token and cryptographic computations take place *inside* the token. Such a token may therefore be viewed as a black-box device. Unfortunately, things are not so easy: they are susceptible to the so-called *side-channel attacks.*

The corresponding threat model is sometimes referred to as *grey-box cryptography.* The adversary has access to the inputs and outputs of the crypto-algorithm *plus* extra side-channel information. The fact that the adversary may have the device in his possession means that he can run a series of experiments and collect information like the running time, the power consumption or the electromagnetic radiation, from which he may infer the secret key. Those attacks are now well understood and efficient (hardware/software) countermeasures are available in recent smart-card implementations.

Grey-box cryptography also encompasses physical attacks (e.g., probing) and fault attacks. In the latter case, an adversary may induce faults and try to recover secret information from the faulty output. Again, there are known protections against those attacks, including sensors (hardware level) and space/time redundancy (hardware or software level).

Note that all the attacks available in the grey-box context are readily applicable — and easier to mount — in the white-box context.

Q7: What are the pros and cons?

Without considering security issues, we list below the advantages of white-box solutions compared to hardware-based solutions. Security aspects are discussed in the next section.

Advantages White-box solutions

– are *cost-efficient*: they are easy to distribute and to install;
– are easily *renewable*: if a security flaw is discovered, updating the software or distributing patches can be done remotely.

Disadvantages White-box solutions

– are orders of magnitude *slower* and require more resources (e.g., memory, processing power, etc);
– are restricted to *symmetric-key* cryptography: there are no known white-box implementations of public-key algorithms.

Q8: How secure are white-box implementations?

There is no complete system that is absolutely secure (and that will never be the case). A system is said secure *relatively* to a security model: one defines the adversary goal (e.g., recovering a key) and the resources the adversary has access to (e.g., oracle decrypting chosen ciphertexts — CCA model).

In the white-box context, it is much more difficult to define the resources of an attacker as they are virtually endless. The best we can hope is to prevent all *known* relevant threats, in an *effective* way. The security is highly dependent on the implementation: there is no need to use strong cryptographic algorithms if they are poorly implemented. Furthermore, white-box implementations are also more sensitive to known attacks; in particular, they are prone to fault attacks.

Concluding Remarks & Open Problems

Most reported attacks exploit *software* security flaws and not weaknesses in cryptographic algorithms. This implies that software protection deserves a higher consideration: *the threats related to the white-box context should be addressed carefully in the design process of secure applications.*

There are currently no fully satisfying solutions for implementing a standard block-cipher in the white-box context (all known proposals are more or less broken). Progresses should be made to provide higher resistance. In some cases, the situation is even worse; we quote two open problems in white-box cryptography:

– the *dynamic* case: when keys are frequently changing over time; and
– the *public-key* case: when public-key techniques are used (e.g., for key exchange, digital signatures, . . .).

In those two cases, there are no known solutions; only token-based schemes are available for building security in the white-box context.

White-box implementations cannot be used *alone* as a protection against key recovery attacks, they should be used in conjunction with other techniques (including non-technical ones). White-box cryptography is still in its early days and requires further research before being widely adopted in commercial products. Grey-box implementations (i.e., token-based solutions) should be preferred — when relevant — as they are more mature: they have a longer history and have undertaken peer evaluation and public scrutiny. Another point against the widespread use of white-box techniques is the penalty cost they may incur: they are orders of magnitude slower and require more resources.[2] On the plus side, we remark that certain techniques of white-box cryptography could be used to improve the security of grey-box implementations against active implementation attacks (e.g., fault attacks). Finally, we remind that security comes at cost and, as a corollary, cannot (should not) be perfect. The *appropriate* security level is dictated by the application (the value of what needs to be protected), the environment (i.e., the threat model) and the costs to develop the corresponding security solution.

Acknowledgments

I am grateful to B. Preneel and B. Wyseur for sending a copy of [7]. I am also grateful to O. Billet, E. Diehl, and C. Salmon-Legagneur for comments.

References

1. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *2001 IEEE Symposium on Security & Privacy*, pages 2–11, IEEE Press, 2001.

[2] This should however be of lesser concern as technologies are evolving. We also admit that the impacts on performance are not that dramatic in certain contexts.

2. J. Bringer, H. Chabanne, and E. Dottax. White-box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468, December 2006. Available at URL http://eprint.iacr.org/2006/468.
3. O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a white-box AES implementation. In *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004.
4. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.
5. S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot. White-box cryptography and an AES implementation. In *Selected Areas in Cryptography – SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2003.
6. S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot. A white-box DES implementation for DRM applications. In *Digital Rights Management – DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2003.
7. J. Cappaert, B. Wyseur, and B. Preneel. Software security techniques. Internal report, COSIC, Katholieke Universiteit Leuven, October 2004.
8. Louis Goubin, Jean-Michel Masereel , and Michael Quisquater. Cryptanalysis of white box DES implementations. Cryptology ePrint Archive, Report 2007/035, February 2007. Available at URL http://eprint.iacr.org/2007/035.
9. M. Jacob, D. Boneh, and E.W. Felten. Attacking an obfuscated cipher by injecting faults. In *Digital Rights Management – DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2003.
10. H.E. Link and W.D. Neumann. Clarifying obfuscation: Improving the security of white-box DES. In *International Conference on Information Technology: Coding and Computing – ITCC 2005*, volume 1, pages 679–684. IEEE Press, 2005. Also available as Cryptology ePrint Archive, Report 2004/025, January 2004 at URL http://eprint.iacr.org/2004/025.
11. A. Main and P.C. van Oorschot. Software protection and application security: Understanding the battleground. International Course on State of the Art and Evolution of Computer Security and Industrial Cryptography, Heverlee, Belgium, June 2003.
12. A.J. Menezes, P.C. van Oorschot, and S.A.Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.
13. T. Sander and C.F. Tschudin. Towards mobile cryptography. In *1998 IEEE Symposium on Security & Privacy*, pages 215–224, IEEE Press, 1998.
14. P.C. van Oorschot. Revisiting software protection. In *Information Security – ISC 2003*, volume 2851 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.
15. Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. Cryptology ePrint Archive, Report 2007/104, March 2007. Available at URL http://eprint.iacr.org/2007/104.