How to Copyright a Function?

[Published in H. Imai and Y. Zheng, Eds., *Public-Key Cryptography*, vol. 1560 of *Lecture Notes in Computer Science*, pp. 188–196, Springer-Verlag, 1999.]

David Naccache[1], Adi Shamir[2], and Julien P. Stern[3]

[1] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
david.naccache@gemplus.com
[2] Weizmann Institute of Science, Applied Mathematics Department
76100 Rehovot, Israel
shamir@wisdom.weizmann.ac.il
[3] UCL Cryptography Group
Bâtiment Maxwell, place du Levant 3, 1348 Louvain-la-Neuve, Belgium
stern@dice.ucl.ac.be
[4] Université de Paris-Sud, Laboratoire de Recherche en Informatique
Bâtiment 490, 91405 Orsay, France
stern@lri.fr

Abstract. This paper introduces a method for tracking different copies of functionally equivalent algorithms containing identification marks known to the attacker. Unlike all previous solutions, the new technique does not rely on any marking assumption and leads to a situation where each copy is either traceable or so severely damaged that it becomes impossible to store in polynomial space or run in polynomial time.

Although RSA-related, the construction is particularly applicable to confidential block-ciphers such as SkipJack, RC4, GOST 28147–89, GSM A5, COMP128, TIA CAVE or other proprietary executables distributed to potentially distrusted users.

1 Introduction

Although software piracy costs $11.2 billion per year [3], impedes job growth and robs governments millions of dollars in tax revenues, most existing protections still rely on legal considerations or platform-specific assumptions.

The most common solutions are based on electronic extensions (dongles) containing memory tables or cheap 4-bit microcontrollers; to rely on these, the protected program periodically challenges the dongle via to the computer's parallel port and makes sure that the retrieved answers are correct. Unfortunately, given enough time, skill and motivation, it is always possible to disassemble the program, find the dongle calls and remove them from the code. In some sense, this approach mixes tamper-resistance and steganography.

A somewhat more efficient solution (mostly used in the playstation industry) consists of executing strategic code fragments in the dongle. As an example, a chess program (exchanging with the player a couple of bytes per round) can be executed in the dongle while less important game parts such as graphics, sounds and keyboard-interfaces can be left unprotected on a CD, useless for playing without the dongle.

A third approach consists of dividing the protected media into two partitions: a first (conventionally formatted) area contains a program called *loader* while the second, formatted in a non-standard way, contains the protected software itself. When the loader is executed, it reads-out the second partition into the RAM and jumps into it. Since operating system commands are unable to read the second partition, its contents are somewhat protected, although patient attackers can still analyze the loader or copy the executable directly from the RAM.

By analogy to the *double-spending problem* met in e-cash schemes, it seems impossible to prevent duplication without relying on specific hardware assumptions, simply because digital signals are inherently copyable. This difficulty progressively shifted research from *prevention* to *detection*, assuming that the former is achieved by non-technical (legal) means. In such models, users generally get personalized yet very similar copies of a given data (referred to as *equivalent*) where the slight dissimilarities (*marks*) between copies are designed to resist collusion, be asymmetric or offer anonymity and other cryptographic features [5, 10, 11].

It is important to stress that all such systems rely on the hypothesis that the marks are scattered in a way that makes their location, alteration or destruction infeasible (*marking assumption*). In practice, marking heavily depends on the nature of the protected data and the designer's imagination [1]. Different strategies are used for source code, images and texts and vary from fractal coding [2], statistical analysis [14] or stereometric image recordings [4] to paraphrasing information exchanged between friendly intelligence agencies [9].

This paper shows that at least as far as functions, algorithms or programs are concerned, marking assumptions can be replaced by regular complexity ones; consequently, we will assume that all identification marks (and their positions) are known to the attacker and try to end-up in a situation where each copy is either traceable or so severely damaged that it becomes impossible to store in polynomial space or run in polynomial time.

The new construction appears particularly suitable to proprietary cryptosystems such as SkipJack, RC4, GOST 28147–89, GSM A5, COMP128 or CAVE TIA, distributed to potentially distrusted users. Although it seems unlikely that an important number (≥ 100) of copies will be marked in practice, we believe that the new method can be useful in the following contexts where a few copies are typically distributed :

– Proprietary standardization committees (such as the TIA-AHAG, the GSM consortium or the DVB group) could distribute different yet equivalent functions to each member-company. Although such a deployment does not in-

criminate individuals, it will point out the company which should be held collectively responsible.
– In an industrial development process, different descriptions of the same function could be given to each involved department (*e.g.* software, hardware, integration and test) and the final client.

Although acceptable, the performances of our solution degrade when the number of users increases; we therefore encourage researchers and implementers to look for new variants and improvements of our scheme.

2 The Formal Framework

The new protocol involves a distributor and several users; the distributor is willing to give each user a morphologically different, yet functionally equivalent, implementation of a function. Hereafter, the word *function* will refer to the mathematical object, while *implementations* will represent electronic circuits or programs that compute a function (more formally, implementations can be looked upon as polynomial circuits that compute the function).

Definition 1. *Let $\mathcal{M}$ and $\mathcal{L}$ be sets of integers. A distribution of the function $f : \mathcal{M} \to \mathcal{L}$ is a set of implementations $\mathcal{F}$ such that:*

$$\forall F \in \mathcal{F},\ \forall x \in \mathcal{M}\quad f(x) = F[x]$$

Definition 2. *Let $\mathcal{M}$ and $\mathcal{L}$ be sets of integers. A keyed distribution of the function $f : \mathcal{M} \to \mathcal{L}$ is an implementation F and a set of integers $\mathcal{K}$ such that:*

$$\forall k \in \mathcal{K},\ \forall x \in \mathcal{M}\quad f(x) = F[x, k]$$

A keyed distribution can be regarded as a monolithic device that behaves like the function f when fed with a key belonging to $\mathcal{K}$, whereas a distribution is simply a set of independent software or hardware devices that behave like the function f. Note that both definitions are equivalent: a keyed distribution is a specific distribution and a keyed distribution can be constructed from a distribution by collecting all the implementations and calling the one corresponding to the key; we will therefore use the simpler definition of keyed distribution.

These definitions do not capture the fact that several implementations might be trivially derived from each other. If, for instance, $F[x, k] = kx$ then it is easy to find an implementation F' such that $F'[x, 2k] = kx$. (F' can be $F[x, 2k]/2$). To capture this, we define an *analyzer*:

Definition 3. *Let $\{F, \mathcal{K}\}$ be a keyed distribution of f. An analyzer $\mathcal{Z}$ of this distribution is an algorithm that takes as input $\{F, \mathcal{K}\}$, an implementation F' of f and tries to find the key $k \in \mathcal{K}$ used in F'. $\mathcal{Z}$ may either fail or output k.*

In other words, when an opponent receives a legitimate implementation of f keyed with k and modifies it, the analyzer's role consists of trying to recover k despite the modifications. The analyzer consequently behaves as a detective in our construction.

2.1 Adversarial model

As usual, the adversary's task consists of forging a new implementation which is unlinkable to those received legitimately. We distinguish two types of opponents: *passive* adversaries which restrict themselves to re-keying existing implementations and *active* adversaries who may re-implement the function in any arbitrary way. When distribution is done through hardware tokens (decoders, PC-cards, smart-cards) where keys are stored in EEPROM registers or battery-powered RAM cells, passive adversaries are only assumed to change the register's contents while active ones may re-design a whole new hardware from scratch.

Definition 4. *Let c be a security parameter. A keyed distribution $\{F, \mathcal{K}\}$ for the function f is c-copyrighted against a passive adversary if given $\mathcal{C} \subset \mathcal{K}$, $|C| < c$, finding a $k \notin \mathcal{C}$ such that $\{F, k\}$ implements f is computationally hard.*[1]

Definition 5. *Let c be a security parameter. A keyed distribution $\{F, \mathcal{K}\}$ with analyzer $\mathcal{Z}$ for f is c-copyrighted against an active adversary if given $\mathcal{C} \subset \mathcal{K}$, $|C| < c$, finding an implementation F' of f such that the analyzer $\mathcal{Z}$, given input F', outputs either a integer k in $\mathcal{K} \setminus \mathcal{C}$ or fails is computationally hard.*

3 The New Primitive

The basic observation behind our construction is that in many public-key cryptosystems, a given public-key corresponds to infinitely many integers which are homomorphic to the secret key, and can be used as such.

For instance, using standard notations, it is easy to see that a DSA key x can be equivalently replaced by any $x + kq$ and an RSA key e can be looked upon as the inverse of any $d_k = e^{-1} \mod \phi(n) + k\phi(n)$. We intend to use this flexibility to construct equivalent modular exponentiation copies.

At a first glance it appears impossible to mark an RSA function using the above observation since given n, e and d_k, a user can trivially find $\phi(n)$ (hereafter ϕ) and replace d_k by some other $d_{k'}$. Nevertheless, this difficulty can be circumvented if we assume that the exponentiation is only a building-block of some other primitive (for instance a hash-function) where e is not necessary.

We start by presenting a solution for two users and prove its correctness; the two-user case will then be used as a building-block to extend the construction to more users.

When only two users are concerned, a copyrighted hash function can be distributed and traced as follows:

Distribution: The designer publishes a conventional hash function h and an RSA modulus n, selects a random $d < \phi$ and a couple of random integers $\{k_0, k_1\}$, computes the quantities $d_i = d + k_i\phi$, keeps $\{\phi, d, k_0, k_1\}$ secret and discloses the implementation $H[x, i] = h(h(x)^{d_i} \mod n)$ to user $i \in \{0, 1\}$.

Tracing: Upon recovery of a copy, the designer analyzes its exponent. If d_0 or d_1 is found, the leaker is identified and if a third exponent d' appears, both users are identified as a collusion.

[1] with respect to the parameters of the scheme used to generate $\mathcal{K}$.

4 Analysis

One can easily show that the essential cryptographic properties of the hash function are preserved and that the distribution is 1-copyrighted against passive adversaries. It seems difficult to prove resistance against general active adversaries; however, we show that if such opponents are bound to use circuits performing arithmetic operations modulo n, then we can exhibit an analyzer that makes our distribution 1-copyrighted.

Theorem 1. *h and H are equally collision-resistant.*

Proof. Assume that a collision $\{x, y\}$ is found in h; trivially, $\{x, y\}$ is also a collision in H; to prove the converse, assume that a collision $\{x', y'\}$ is found in H. Then either $h(x')^d = h(y')^d \mod n$ and $\{x', y'\}$ is also a collision in h, or $h(x')^d \neq h(y')^d \mod n$ and $\{h(x)^d \mod n, h(y)^d \mod n\}$ is a collision in h. □

Lemma 1. *Finding a multiple of $\phi(n)$ is as hard as factoring n.*

Proof. This lemma, due to Miller, is proved in [6]. □

Theorem 2. *If factoring is hard, $\{H, \{d_0, d_1\}\}$ is 1-copyrighted against a passive adversary.*

Proof. Assume, without loss of generality, that an adversary receives the implementation H and the key d_0. Suppose that he is able to find $d' \neq d_0$ such that $H[., d_0] = H[., d']$. Then, $d_0 - d'$ is a multiple of $\phi(n)$ and by virtue of Miller's lemma, n can be factored. □

Theorem 3. *If factoring is hard, then $\{H, \{d_0, d_1\}\}$ is 1-copyrighted against an active adversary restricted to performing arithmetic operations modulo n.*

Proof (Sketch). We show that an active adversary is not more powerful than a passive one. We build $\mathcal{Z}$ as follows: $\mathcal{Z}$ first extracts the exponentiation part. He then formally evaluates the function computed by this part, with respect to its constants $\{c_1, \ldots, c_w\}$ and input x, replacing modular operations by regular ones. This yields a rational function P/Q with variable x and coefficients depending only on $\{c_1, \ldots, c_w\}$. He finally evaluates all these coefficients modulo n. A careful bookkeeping of the non zero monomials shows that either the adversary has obtained a multiple of $\phi(n)$ (and can therefore factor n) or that P divides Q. This means that the rational function is in fact reduced to a single monomial, from which we can compute the value of the corresponding exponent and the security of the construction follows from the security against the passive adversary. □

Note that resistance against active adversaries is more subtle than our basic design: assuming that d is much longer than ϕ, adding random multiples of ϕ to d will not alter its most significant bits up to a certain point; consequently, there is a finite number of ℓ-bit exponents, congruent to $d \mod \phi$ and having a given bit-pattern (say u) in their most significant part; the function: $h(h(x)^{d_i} \oplus u)$ will thus admit only a finite number of passive forgeries.

5 Tracing More Users

Extending the previous construction to more users is somewhat more technical; obviously, one can not simply distribute more than two exponents as this would blind the collusion-detection mechanism.

System setup is almost as before: letting t be a security parameter, the designer publishes a hash function h and t RSA moduli $\{n_1, \ldots, n_t\}$, selects t random triples $\{d[j] < \phi_j, k[0,j], k[1,j]\}$ and computes the t pairs:

$$d[i,j] = d[j] + k[i,j]\phi_j \quad \text{for} \quad i \in \{0,1\}$$

Then, the designer selects, for each user, a t-bit string ω. We will call ω the *ID* or the *codeword* of this user. Each user receives, for each j, one out of the two keys $d[0,j], d[1,j]$ (he receives $d[0,j]$ if the j-th bit of ω is zero and $d[1,j]$ otherwise). The exact codeword generation process will be discussed later.

Let s be a security parameter ($0 < s \leq t$). The function is now defined as follows: the input x is hashed and the result $h(x)$ is used to select s keys among the t keys of a user. For simplicity, let us rename these s keys $\{a_1, \ldots, a_s\}$ for a given user.

We now define $H[x] = H[s,x]$ recursively by:

$$\begin{aligned} H[1,x] &= h(x^{a_1} \mod n_1) && \text{and} \\ H[j,x] &= h(H[j-1,x]^{a_j} \mod n_j) && \text{for } j > 1 \end{aligned}$$

A simple (and sometimes acceptable) approach would be to distribute copies with randomly chosen codewords. However, by doing so, logarithmic-size collusions could recover the exponents with constant probability and forge new implementations; therefore, specific sets of codewords must be used. Letting $\mathcal{C}$ be a coalition of c users provided with codewords $\omega[1], \ldots, \omega[c]$. $\mathcal{C}$ can not change $d[i,j]$, if and only if all codewords match on their j-th bit. Hence, the problem to solve boils down to the design of a set of codewords, amongst which any subset, possibly limited to a given size, has elements which match on enough positions to enable tracing. This problem was extensively studied in [4] which exhibits a set of codewords of polylogarithmic ($\mathcal{O}(\log^6 t)$) length, capable of tracing logarithmic size coalitions.

While [4]'s hidden constant is rather large, our marks are a totally independent entity and their size is not related to the size of the function (which is *not* the case when one adds marks to an image or a text); hence, *only complexity-theoretic considerations* (the hardness of factoring n) may increase the number of symbols in H.

Finally, the new construction allows to adjust the level of security by tuning s accordingly. Although pirates could try to distribute copies with missing exponents in order not to get traced, such copies become almost unusable even if only a few exponents are omitted. This approach (detecting only copies which

are usable enough) is similar to the one suggested by Pinkas and Naor in [8]. Assuming that m of the exponents are missing and that each computation requires s exponents out of t, the correct output probability is:

$$\Pr[t, m, s] = \frac{(t-s)!(t-m)!}{t!\ (t-m-s)!}$$

Given $\Pr[t, m, s]$'s quick decay (typically $\Pr[100, 10, 10] \cong 3/10$) and the fact that repeated errors can be detected and traced, it is reasonable to assume that these untraceable implementations are not a serious business threat. No one would buy a pirate TV decoder displaying only three images out of ten (the perturbation can be further amplified by CBC, in which case each error will de-synchronize the image decryption until the next stream-cipher initialization).

6 Applications

Building upon a few well-known results, a variety of traceable primitives can be derived from H: Feistel ciphers can be copyrighted by using H as a round function, traceable digital signatures can use H in Rompel's construction [13] and traceable public-key encryption can be obtained by using [7] with a composite modulus (e-less RSA) or by post-encrypting systematically any public-key ciphertext with a watermarked block-cipher keyed with a public constant. Interactive primitives such as zero-knowledge protocols or blind signatures can be traced using this same technique.

The construction also gives birth to new fundamental protocols; a web site could, for example, sell marked copies of a MAC-function and record in a database the user IDs and their exponents. Since all functions are equivalent, when a user logs-in, he does not need to disclose his identity; but if an illegitimate copy is discovered, the web owners can look-up the faulty ID in the database[2].

Another application consists of restricting software to registered users. In any scenario involving communication (file exchange, data modulation, payment, etc), the protected software must simply encrypt the exchanged data with a copyrighted block-cipher. Assuming that a word processor systematically encrypts its files with a copyrighted block-cipher (keyed with some public constant), unregistered users face the choice of getting traced or removing the encryption layer from their copies (the word processor will then be unable to read files produced by legitimate users and will create files that are unreadable by registered programs); consequently, users of untraceable (modified) programs are forced to *voluntarily* exclude themselves from the legitimate user community.

Finally, our scheme can also be used for TV tracing instead of the usual broadcast encryption/traitor tracing techniques. In broadcast schemes, the message is usually block-encrypted, each block being made of a *header* (which allows each user to recover a random key) and a *ciphertext* block (which is the encryption of the data under this random key). The main advantage of our scheme is its

[2] care should be taken not to restrict the MAC's input space too much as polynomially small I/O spaces could be published as look-up tables.

very low communication overhead: the header can be a simple encryption of the secret key, as all the users receive an equivalent decryption function. There are, however, several disadvantages: we totally lose control over the access structure allowed to decrypt. This means that new keys need to be sent to all registered users from time to time.

Surprisingly, in our setting smart-cards suddenly become a powerful... piracy tool; by programming one of the H_i into a smart-card, a pirate can manufacture and distribute executable hardware copies of his function and rely on the card's tamper-resistance features to prevent the designer from reading the exponents that identify him.

7 Conclusion and Open Questions

We presented a new (public-domain) marking technique which applies to a variety of functions and relies on regular complexity assumptions; while we need a large amount of data to personalize an implementation when many users are involved, the construction is fairly efficient and can be adjusted to variable security levels.

There remains, however, a number of fundamental and practical questions such as the existence of DLP-based copyright mechanisms or the design of a copyright mechanism that allows to serve more than two users in a single (non-iterated) function. From a practical standpoint, it seems easy to compress the set $\{n_1 \dots n_t\}$ to only $N + t \log N$ bits (this is done by generating t moduli having identical MSBs). Reducing the size of the exponent set is an interesting challenge.

References

1. J.-M. Acken, *How watermarking adds value to digital content*, Communications of the ACM, vol. 41-7, pp. 75-77, 1998.
2. P. Bas, J.-M. Chassery and F. Davoine, *Self-similarity based image watermarking*, Proceedings of EUSIPCO'98, Ninth European signal processing conference, European association for signal processing, pp. 2277-2280.
3. *The huge costs of software piracy*, Computer Fraud and Security Bulletin, 09/1997, Elsevier Science, page 3.
4. D. Boneh and J. Shaw, *Collusion-secure fingerprinting for digital data*, Advances in cryptology CRYPTO'95, Springer-Verlag, Lectures notes in computer science 963, pp. 452–465, 1995.
5. B. Chor, A. Fiat and M. Naor, *Tracing traitors*, Advances in cryptology CRYPTO'94, Springer-Verlag, Lectures notes in computer science 839, pp. 257–270, 1994.
6. G. Miller, *Riemann's hypothesis and tests for primality*, Journal of computer and system sciences, vol. 13, pp. 300–317, 1976.
7. D. Naccache and J. Stern, *A new public-key cryptosystem*,Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 27–36, 1997.

8. M. Naor and B. Pinkas, *Theshold Traitor Tracing*, Advances in cryptology CRYPTO'98, Springer-Verlag, Lectures notes in computer science 1462, pp. 502–517, 1998.
9. V. Ostrovsky, *The other side of deception*, Harper-Collins Publishers, New-York, page 38, 1995.
10. B. Pfitzmann and M. Schunter, *Asymmetric fingerprinting*, Advances in cryptology EUROCRYPT'96, Springer-Verlag, Lectures notes in computer science 1070, pp. 84–95, 1996.
11. B. Pfitzmann and M. Waidner, *Anonymous fingerprinting*, Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 88–102, 1997.
12. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120-126, 1978.
13. J. Rompel, *One way functions are necessary and sufficient for secure digital signatures*, Proceedings of the 22-nd Annual ACM Symposium on the Theory of Computing, pp. 387–394, 1990.
14. K. Verco and M. Wise, *Plagiarism à la mode: a comparison of automated systems for detecting suspected plagiarism*, The Computer Journal, vol. 39–9, pp. 741–750, 1996.