

“VALIDY TECHNOLOGY” WHITE PAPER

SOLUTION AGAINST SOFTWARE PIRACY
AND IT SABOTAGE



IT AND ECONOMIC SECURITY

www.validy.com / www.validy-licensing.com

TABLE OF CONTENTS

page 1	1. State of the art in software piracy protection
	1.1 Simple additive protections
	1.2 Additive protections with material device
	1.3 Subtractive protections
page 2	2. State of the art in protection against IT sabotage
	2.1 Verification of integrity before the execution starts
	2.2 Verification of integrity by executing the program in a device
	2.3 Verification of integrity by parallel execution
page 3	3. Validy's invention
	3.1 The four criteria for a subtractive method that works
	3.2 The execution protection system that abides by those four criteria
page 6	4. Strong additional benefits
	4.1 "Tag" instructions identification system
	4.2 Mutual protection
	4.3 Additive calculations, subtractive calculations.
	4.4 Locking of additive calculations
	4.5 Ultimate reprisals
	4.6 Integrity of the program
	4.7 The data protection sytem
	4.8 Measuring usage
	4.9 Limiting functionalities
	4.10 Rigidity or flexibility of the device
	4.11 Fixed or detachable device
page 10	5. Developing programs
	5.1 Implications of the use of "Validy Technology" on developing
	5.2 Protection against piracy: no library
	5.3 Debug process
	5.4 Integrity protection
	5.5 Validy compiler
	5.6 Other compilers
	5.7 Additional tools
page 11	6. Implications on deployment
	6.1 Various supports
	6.2 Production
	6.3 Maintenance

I • STATE OF THE ART IN SOFTWARE PIRACY PROTECTION

S

I•1 Simple additive protections

Additive protections consist in verifying execution rights for instance by testing the presence of a CD-Rom or a serial number on the computer or on the Internet, ...

Those tests are added to the program by the developer. They are not indispensable to the functioning of the program. They can easily be spotted and understood by the pirate, who can then either reply correctly or bypass them.

To make things harder for pirates, developers sometimes use techniques known as "obscuration techniques" that attempt to hide the additions.

Despite those techniques, determined pirates always end up finding and bypassing the protection.

If just one pirate publishes the results of his research on the Internet under the form of an automatic program (called "crack"), any Internet user become capable of bypassing the additive protection.

I•2 Additive protections with material device

To make protection stronger, some companies have designed devices called "dongles".

Those devices can take different forms: cap on the parallel port of the computer, smart card, USB key, integrated circuit, ... They often use cryptography techniques that prevent the pirate from replying correctly to the tests. Despite that, the pirate, even if he can't reply correctly to the tests, always ends up managing to bypass them.

I•3 Subtractive protections

Subtractive protections consist in hiding part of the program into a device in order to remove that part from the pirate's view. That device can also take different forms: cap on the parallel port of the computer, smart card, USB key, integrated circuit, ...

Classic subtractive protections consist in remoting functions to the device. The process goes as follows: the parameters for a function are supplied to the device by the protected program, the function is executed by the device and the results of that function are returned to the protected program (system known as "Remote Procedure Call").

The difficulty consists in finding the "right" functions to remote to the device. The pirate can indeed bypass the protection using several methods:

If the remoted function is not indispensable to the program's execution, the pirate can simply suppress it while conserving the major part of the program's functionality.

If the remoted function often uses the same values as parameters, the pirate, after having analyzed for some time the exchanges between the protected program and the device, is able to write a program that imitates the device and replies in lieu of it, even if he does not understand the remoted function.

If the remoted function can be identified, the pirate, after having analyzed for some time the exchanges between the protected program and the device, is able to write a program that imitates the device and simulates the remoted function, whatever the parameters are.

In the three cases above, with some persistence, the pirate is able to modify the protected program sufficiently so that it works without the device.

To make up for those weaknesses, the developer is then led to remote, if possible, complicated functions. Such practice is limited for two reasons: many programs do not incorporate complicated enough functions, and when they do, executing said functions in the device, which is much slower than the computer, considerably slows down the protected program, which quickly becomes unusable.

2•STATE OF THE ART IN PROTECTION AGAINST IT SABOTAGE

V

2•1 Verification of integrity before the execution starts

Verifying the integrity of a program before it starts consists in using an auxiliary program which checks that the program to execute has not been modified.

Such verification has two flaws that can be exploited by a pirate:

The pirate can modify the program after the verification and before the execution.

The pirate can modify the auxiliary verification program so that it accepts modified programs.

2•2 Verification of integrity by executing the program in a device

When a program is entirely executed in a device, the device can verify itself in a secure manner the program's integrity before it starts, hence eliminating the two weaknesses of the previous point. This protection is very effective but considerably limits the size and performance of the protected program that must entirely execute in the device, that is much smaller and slower than the computer (slows down the program by at least two orders of magnitude).

2•3 Verification of integrity by parallel execution

When great reliability is required, a given program or similar programs can be executed in parallel on several machines. On a regular basis, the various machines mutually check that the progresses of the programs are similar and in case a problem is detected, a decision to correct it is taken. Modifying the program's integrity therefore requires having access to all the machines and modifying all the programs. Such approaches are very complex and costly and reserved to very particular applications (aeronautics, nuclear plants, ...).

3 • VALIDY'S INVENTION



3•1 The four criteria for a subtractive method that works

As previously tackled in the paragraph that describes subtractive protections, the part of the program hidden in a device must abide by a certain number of criteria. For the protection to be effective and usable, four fundamental criteria must be abided by:

- **The hidden part must be essential:**

One must not add a “decorative” part to the program with the intention of hiding it in the device. Indeed, the pirate can then simply suppress it while conserving the major part of the program’s functionality.

For instance, if the hidden part is only used for calculating a window background, the program will have most of its functionality even if the background is not displayed.

- **The hidden part must be non simulable:**

By analyzing the exchanges between the protected program and the device, the pirate must not be able to understand the functioning of the part of the program that is executed in the device. Indeed, understanding it would enable him to write a simulator program to get rid of the device.

For instance, if the hidden part carries out a sine function, the pirate can identify and simulate it.

- **The hidden part must be non replayable:**

By analyzing for some time the exchanges between the protected program and the device, the pirate must not be able to imitate the device’s responses. Indeed, imitating the device, even without understanding the meaning of the exchanges, would enable him to write an imitation program to get rid of the device.

For instance, if the exchanges between the protected program and the device are repeated from an execution to another, the pirate can record the responses during a first execution and “replay” them with his imitation program during the subsequent executions.

- **The hidden part must not slow down the execution:**

For a subtractive protection to be usable, the slowing down it introduces in the protected program’s execution must be reasonably low. Indeed, the difference in performance between the processor executing the non-hidden part of the protected program and the device executing the hidden part of the protected program is often very important (two orders of magnitude or more). Judiciously choosing the hidden part is therefore essential.



3 • VALIDY'S INVENTION



3•2 The execution protection system that abides by those four criteria

The six facets of the invention:

• Variable:

The first facet of the invention consists of causing variables to reside in the hardware device during the execution of a protected program.

When allocating a value to a variable residing in the device, said value is transferred to the device.

When using a variable residing in the device, its value is transferred to the computer.

During their stay in the device, the values of the variables are not visible to a pirate.

This facet introduces a protection by the fact that part of the program's state resides in the device and that consequently, its presence is indispensable to the functioning of the program.

However this facet used alone is not sufficient to ensure effective protection as a determined pirate can observe and understand the exchanges between the computer and the device, and make a simulator of the latter.

• Temporal dissociation:

The second facet of the invention consists of modifying the value of the variables remoted in the hardware device. These modifications are carried out independently of the moments of the transfers from or to the device, they are triggered by commands sent by the computer to the device.

This facet strengthens the protection as it makes the exchanges between the computer and the device more complex by decorrelating the transfers from the commands of modification of each variable and the variables between themselves.

However this is not always enough to ensure effective protection as despite the complexity of the exchanges, a determined pirate can always observe, modify and understand the exchanges between the computer and the device and make a simulator of the latter.

• Elementary functions:

The third facet of the invention consists of breaking down the functions modifying the variables in the hardware device into elementary functions. These elementary functions are triggered by elementary commands sent from the computer to the device. They are of the type ADD, MOVE, ...

This facet enables to standardize the dialogue using a set of elementary functions in the device so as to synthesize any function.

This facet strengthens even more the protection as it removes the limitations in number and size of the functions that can be processed in the device. Moreover, the elementary commands enabling to synthesize different functions can be intertwined to make the dialogue even more complex.

However this is still not sufficient to ensure effective protection as despite the increased complexity of the dialogue, a determined pirate can still observe, modify and end up understanding the meaning of the elementary commands and make a simulator.

3 • VALIDY'S INVENTION

R

- **Renaming:**

The fourth facet of the invention consists of renaming the elementary commands exchanged between the computer and the hardware device so as to make them unintelligible. The renaming is carried out by encrypting the elementary commands during the protection of the program. During execution, the device decrypts the renamed elementary commands it receives and executes the triggered elementary functions.

This facet strengthens the protection because the pirate can never access the elementary commands in their non-encrypted form and therefore cannot analyze their structure. Neither can he generate new elementary commands to send requests to the device and try to understand the responses to these requests.

At this point, the protection is already very solid and beyond the capacity of most pirates. The last possible attack consists in trying to apprehend the elementary functions executed in the device by making assumptions on their meaning and testing those assumptions by modifying the order of execution of those functions.

- **Detection and coercion:**

The fifth facet of the invention consists of verifying that the part of the program executed in the hardware device is executed in a manner that conforms to what was provided for during the phase of protection of the program (compilation). To this end, the device must be considered as a security coprocessor executing a set of security instructions. These security instructions include additional fields enabling to control that the chaining of their executions is in accordance with what was planned. If the chaining is not as expected, the security coprocessor immediately detects it and takes the coercion measures decided by the developer. As the protected program cannot work without the coprocessor, it has real retaliatory power since it can decide to stop working, hence stopping the program.

This facet prevents the last possible attack, which consists in modifying the order of execution of the instructions to try to understand them.

At this point, the pirate cannot touch to the exchanges between the computer and the device without being immediately detected.

- **Conditional branch:**

The sixth and last facet of the invention consists of concealing the structure of the program by making use of the hardware device to provide branching addresses.

This is achieved by making state variables that condition the program's execution reside in the device, carrying out in the device the calculations enabling to know the branching address and finally returning that address to the computer.

This facet completes the protection by hiding from the pirate's view the branching decisions and the reasons for those decisions, which takes the implementation logic of the program away from him.

4•STRONG ADDITIONAL BENEFITS



4•1 “Tag” instructions identification system

The most effective manner of implementing the “detection and coercion” invention is to use an instruction identification system that uses labels called “tags”.

These “tags” are integral parts of the operation code of each instruction and of the coprocessor resources.

In each instruction, there is a tag to identify that instruction and one or several other tags to identify the expected source of each operand. The coprocessor registers can store not only data, but the tag identifying the instruction that generated the data as well.

With that identification system, the coprocessor is able to verify during the protected program’s execution that the chaining of its instructions has not been modified.

An instruction can for instance be of the following form:

ADD R1 <t1> R2 <t2> R3 <t3>

Which means: Check that R2 has been produced by an instruction identified by the tag t2 and that R3 has been produced by an instruction identified by the tag t3. Report possible errors to the coercion system. Add R2 and R3 data and store the result in the data part of R1. Also store the value t1 as instruction identifier in the tag part of the register R1.

Any compiler has dependency information enabling to easily calculate the tags necessary for this invention.

In the phase of protection of the program, the Validy compiler automatically generates the tags and insert them in the instructions of the security coprocessor.

4•2 Mutual protection

The detection and coercion system very simply enables to make the successful completion of the execution of a calculation in the device depend on the prior execution of another calculation in the device.

The method used to create this dependency is to execute the two following security instructions:

- generate the value 0 by subtracting the result of the first calculation from itself
- add that 0 to the result of the second calculation

The chaining verification system explained above then makes the second calculation fail if the first calculation has not been carried out or if the two added security instructions have been modified or deleted.

4•STRONG ADDITIONAL BENEFITS

M

More specifically:

Let's suppose that R1 <t1> is the result of calculation1 with the tag t1 and R2 <t2> is the result of calculation2 with the tag t2.

and let's carry out:

```
SUB R3 <tempt> R1 <t1> R1 <t1>
```

```
ADD R2 <newt2> R2 <t2> R3 <tempt >
```

The first instruction generates a 0 tagged with tempt in the register R3 if calculation1 has been carried out, and fails otherwise.

The second instruction generates a new tag for R2 without changing its data, and fails if the first instruction has been modified or deleted.

In the end, the result of calculation2 is available in R2 with the new tag newt2 only if calculation1 has been carried out.

Consequence:

This dependency system, applied several times, enables the mutual protection of 2 calculations or more.

4•3 Additive calculations, subtractive calculations

An additive calculation is a calculation that is added to the program to protect and that is carried out in the device. As it's an addition to the program, an additive calculation is not indispensable.

A subtractive calculation is a calculation that is part of the program to protect and that is subtracted from the pirate's view by carrying it out in the device. As it's part of the program, a subtractive calculation is indispensable.

4•4 Locking additive calculations

The mutual protection described above enables to link calculations together, be they additive or subtractive.

It is then possible to add additive calculations to the program and to create a web of links that is built on a "base" of subtractive calculations and that links all the additive calculations to that base.

4•5 Ultimate reprisals

When all the calculations are linked together, any attempt to modify or delete any one or several of the calculations is immediately detected by the device that then takes the appropriate coercion measures.

The only solution for the pirate then consists in simultaneously removing all the calculations carried out in the device. The existence of the "base" of subtractive calculations unknown by the pirate and indispensable to the program forces him to reinvent all those calculations simultaneously which is impossible!

4•STRONG ADDITIONAL BENEFITS



4•6 Integrity of the program

The additive calculations added can be used to verify the correct functioning of the program and thus defend its integrity.

Those calculations never require any transfer of information from the device to the computer and are therefore very effective, they can for instance take the following forms:

- verification of the call graph of a subprogram
- verification of the number of iterations of a loop
- verification of the time elapsed to execute a function, to prevent step by step execution
- etc.

4•7 The data protection system

The additive calculations added can be used to protect the data of the program and in particular its exchange data.

For instance, the writing of encrypted data into a database with impossibility for the pirate to compromise the writing or reading of this data and to modify the database's data.

4•8 Measuring usage

The additive calculations added can be used to measure the use of various functionalities of the program such as for instance the usage time, the number of pages printed, the number of files saved, ... and thus enables to forbid their use beyond a certain threshold. This enables selling software depending on its usage (pay per use / pay as you go).

4•9 Limiting functionalities

The additive calculations added can be used to limit the functionalities of the program depending on the state of the device.

For instance: depending on the state of the device, the same program will either be an evaluation version with reduced functionalities, or a commercial version with all functionalities.

4•STRONG ADDITIONAL BENEFITS

R

4•10 Rigidity or flexibility of the device

Depending of its applications, a software publisher can decide to use a rigid or a flexible device.

If the device is rigid, no software update is possible without changing device. This ensures that a certified version of a program is never modified. In this case the device can be considered as a seal on the software. For instance a program used to control a car braking system.

If the device is flexible, the software updates are possible without changing device, which enables to install patches and minor updates without constraint for the user. This flexibility is a significant advantage for most software editors.

4•11 Fixed or detachable device

Depending on applications, a software editor can decide to use a fixed or detachable device.

If the device is fixed, i.e. permanently installed in a machine, the software can be executed only in that machine. For instance, in the case of an embedded software working on a dedicated machine that cannot be moved to another one for quality control or security reasons.

If the device is detachable, it can be used on different machines. Only the machine that has the device connected to it can execute the software. It would allow for instance to use the same program at work and at home, provided that the user carries the device with him.

5 • DEVELOPING PROGRAMS

5•1 Implications of the use of “Validy Technology” on developing

“Validy Technology” introduces minimum changes during the writing or the protection of a program. The Validy compiler takes care of almost all the modifications.

5•2 Protection against piracy: no library

To protect a program against piracy, “Validy Technology” requires no call of function belonging to a security library. The only modifications consist in choosing and marking the variables that must reside in the device.

In concrete terms, it only takes to insert a comment line before the declaration of each chosen variable. The Validy compiler then takes care of generating the security instructions necessary to the implementation of the various facets of “Validy Technology”.

5•3 Debug process

Once all the variables have been chosen and marked, the developer begins by compiling his program with his usual compiler and carries out the debugging with his usual debugger.

Once he is satisfied with the result, the developer compiles the program with the Validy compiler and carries out the debugging with a debugger and a simulator of the device. He thus checks that the protected program works correctly.

Finally, the developer compiles the final version of his program with the Validy compiler and tests the definitive version with a Validy device.

NB: During the final compilation, the renaming is carried out using a compilation device. The developer has therefore no access to the renaming function, which prevents any leak at that level.

5•4 Integrity protection

The vast majority of the integrity protection of a program is done automatically by the Validy compiler: For instance, an automatic check of the call graph is carried out by automatically adding security instructions that control the structure of the program.

In case specific additional protection is needed, a library is at the developer's disposal.

5•5 Validy compiler

So as to be independent of the supported platforms, Validy has developed a compiler that works at the byte code level. The Validy compiler takes Java or CIL (.NET platform's intermediate language) byte code as source, adds the “Validy Technology” protection and generates respectively Java or CIL byte code.

The Validy compiler enables to support the following languages: Java, C#, Visual Basic .Net, C++ .Net, ... on all machines supporting those languages.

5•6 Other compilers

Validy is considering developing a version of gcc that adds the “Validy Technology” protection.

Validy is also considering developing a Pentium binary recompiler to protect a program, from its executable code.

5•7 Additional tools

Validy will develop additional tools to make implementing the protection easier and in particular a profiling tool that helps choosing variables or automatically selects them.



6•IMPLICATIONS ON DEPLOYMENT



6•1 Various supports

The “Validy Technology” protection necessarily requires using a secure device, that can take various forms: USB key, smart card, electronic component, ...

USB keys and smart cards are detachable whereas the electronic components are fixed (for their respective benefits see § 4.1.1).

The secure devices are supplied by Validy and called VSMs (Validy Security Modules). They incorporate the latest security features and in particular counter-measures to counter SPA, DPA, DEMA, and SEMA attacks.^(*)

6•2 Production

Client VSMs have to be customized from information used during compilation and contained in the master VSM (see § 5.3). Customization is totally independent of Validy. Validy does not have access to any of that information.

Logistics and means implemented for customization may vary depending on the production context: For small quantities, information is transferred directly from the master VSM to client VSMs.

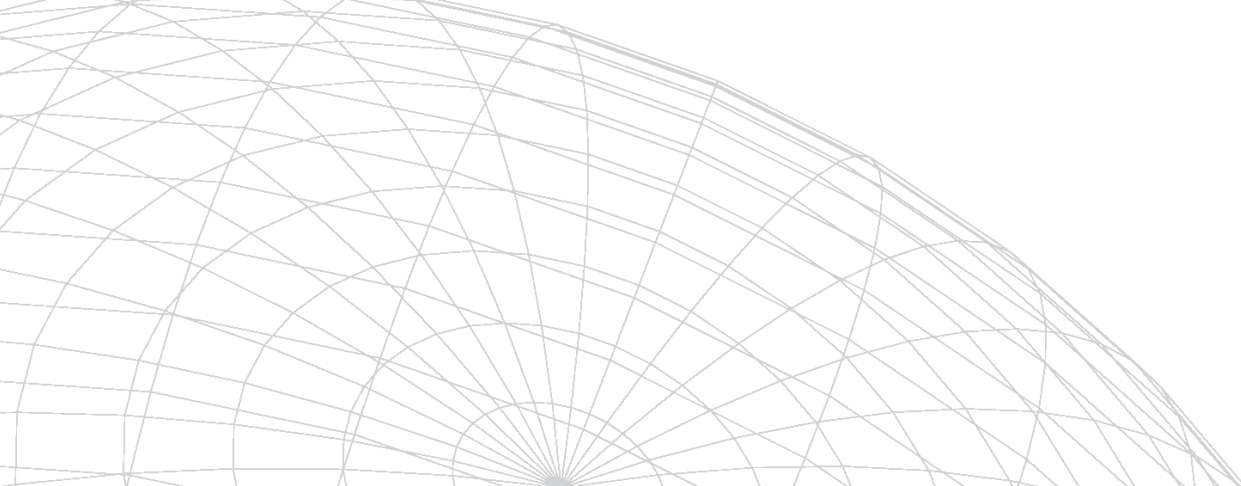
For large quantities, information is transferred from the master VSM to production VSMs. These production VSMs, located on the production machines or distant, transmit in their turn the information to the client VSMs, locally or via the Internet.

The implementation of standard cryptographic techniques enables to securely transfer the information between the various VSMs, without risk of leak, and whatever the distance.

6•3 Maintenance

The maintenance mode is specific to each business model. This subject is tackled in the white paper on business models.

^(*): SPA: Simple Power Analysis, DPA: Differential Power Analysis, DEMA: Differential ElectroMagnetic Analysis, SEMA: Simple ElectroMagnetic Analysis



SA Validy
ZI - 5, rue Jean Charcot
26100 Romans - France
info@validy.com



Validy Net Inc.
1001 SW Fifth Avenue
suite 1100
Portland, OR 97204 - USA