Patrick Wolf, Martin Steinebach, and Sascha Zmudzinski
Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstr. 75, 64295 Darmstadt, Germany
[patrick.wolf | martin.steinebach | sascha.zmudzinski]@sit.fraunhofer.de

ADAPTIVE SECURITY FOR VIRTUAL GOODS - BUILDING AN ACCESS LAYER FOR DIGITAL WATERMARKING

Abstract: Digital watermarking has become an accepted media security technology used in diverse application areas from copyright and integrity protection to broadcast monitoring. Watermarking algorithms are similarly diverse. They vary in media type, required knowledge (i.e. parameters) and implementation language. These differences make it hard for application developers to integrate watermarking into their applications and even harder to maintain and upgrade applications already incorporating watermarking.

This paper proposes an approach to provide generic access to watermarking functionality through a light-weight framework called *AlgorithmManager*. It is independent from any algorithm-specific properties like implementation language, medium type or required parameters, so that also algorithms implemented in C/C++ become usable within the framework.

Keywords: Usage of Digital Watermarking, AlgorithmManager

1. Motivation

Digital watermarking has become the security mechanism of choice for many novel distribution platforms of multimedia content. The great flexibility coming with watermarked goods and the customer friendliness derived from this, have caused a wide acceptance of this technology. Watermarking solutions used in B2C multimedia distribution are usually so-called transactional watermarks, embedding on the fly transaction codes for customer identification.

1.1 Challenges

From the wide acceptance new challenges have risen. One is the required high embedding speed, solved e.g. by preprocessing [1] within watermarking ner solutions [10]. Another challenge is the flexible integration of the narking technology within the distribution platforms. As the market is in movement these days, new approaches for selling, renting and distributing virtual multimedia goods come and go. Lengthy integration times of security measures are often inacceptable in an industry building shop solutions on Open Source platforms offering most of the required functionality with minimal changes.

Digital watermarking has three characteristics which make it hard to use it in a similar manner:

- Watermarking solutions are usually implemented in C code to ensure good processing times
- Watermarking requires parameter optimization for different scenarios and content types
- The development of watermarking algorithms has not yet come to an end, new solutions are proposed within comparatively short time

So, when integrating watermarking embedding and/or detection functionality into applications, programmers often find that they have to deal with many algorithm-specific details before their application actually can do ”watermarking”. Such details include the algorithm’s commandline syntax or its library interface, respectively, the type or even the format of media to be watermarked, required technical parameters like watermark-strength and length of the message or such essential issues as the representation of the watermark message or the secret watermark key: Is it binary or a string?

Such simple but important details are obstacles for watermarking in becoming a black box principle; black box meaning that for using watermarking no further knowledge about the interior is needed. ”Encryption” is such an example of a successful black box: It is a widely known principle, required parameters are mostly clear and it can be easily integrated into applications. Even though watermarking is fundemantally different from encryption [2], the AlgorithmManager framework described in this work, should be seen as a (first) step into the same direction and should pave the way for watermarking a more widely used security technology.

This calls for a middle layer offering the following functionality to potential users of watermarking algorithms or respectively their technical staff setting of the content distribution platforms.

- Java-based Open Source for transparency and easy adaption into existing frameworks or software solutions
- Independence from individual watermarking algorithms to simplify exchange of algorithms when necessary
- Flexible watermarking parameter handling allowing easy adaption of new parameter sets and options coming with new versions of watermarking algorithms

In the following sections we described an approach providing such a middle layer already successfully applied in customer solutions connecting C-based watermarking code with Java-based business scenarios.

1.2 Use case

To provide a more concrete perspective on the potential impact of a unified way to deal with watermarking algorithms, we discus application scenarios for the audio watermarking algorithm provided by our institute.

The most relevant applications include:

- Customer identification watermarking for online shops selling music or audio books to end consumers.
- Copyright watermarking for online shops selling music or audio books to end consumers or for public audio archives preserving the cultural heritage.
- User identification for promotional copies distributed via Internet as mp3 downloads or as individually marked CDs as a business to business service.
- Watermarking sound tracks for Video-on-Demand systems for the identification of distribution channels or users.
- Automated Internet search mechanisms for copyright infringements of any type of watermark protected content listed before.

All these scenarios use the same basic watermarking algorithm but need to integrate into a multitude of different business environments. For example, online shops usually integrate watermarking technology either in the content management system or the web content delivery service. Watermarking promotional CDs requires the integration of the watermarking process into the CD-duplication-system. In all these cases, various operating systems and software environments may be found as a host for the watermarking algorithm. This includes Java, C-derivates, scripting languages and Delphi, running on Linux, Windows and Macintosh systems.

Sometimes several audio watermarking embedding solutions from different institutions or watermarking companies have to integrated because online shop service providers or manufacturers of CD duplication devices may incorporate watermarking soultions from different partners. This applies even more for the related watermarking detection solutions integratied in Internet search systems for copyright infringements[11]: Here, the clients from the content industry, e.g. music labels, publishers or movie studios, usually have use different partners for both the distribution and the watermark protection.

The same holds for the problem of evaluating and *benchmarking* of different watermarking solutions from different instututions and companies with respect to sound quality anf robusness and security against attacks [6]. One example is the so-called Stirmark benchmark audio watermarking [9]. For such an evaluation system external watermarking algorithms have to be integrated.

Without a middleware, every time a new environment wants to access the algorithm, new interface structures may be necessary. With the help of such a middleware, only the core algorithm needs to be transferred to the different operating systems, calling the algorithm by the application is then done by the middleware which may be platform-independent and transparent. This drastically reduces the efforts and therefore the cost of integrating the watermarking algorithm within user systems. Even more when the customer may be interested in switching between watermarking algorithms at a later point as may be the case in the video on demand environment where the user may want to apply video watermarking when this technology is more advanced.

2. Framework Design

The goal of the AlgorithmManager is to take an arbitrary watermarking algorithm, copy its implementation into a directory, fill out a configuration file and be ready to use it in any application. Applications should be freed from the burdens each individual implementation of watermarking functionality brings them.

The first step is therefore to condense watermarking functionality into a generic set of interfaces that still allow individual configuration of algorithms. This results in the definition of an embedder interface that takes a cover medium, a watermark message and a secret key and returns a watermarked medium as described many time in the literature (e.g. in [3], [8]). Detection reads a watermark message by taking a medium to be analyzed and the key as parameters. This is basically all an application needs to know about watermarking.

The hard part about watermarking is the implemenation of algorithms that do this and satisfy all necessary imperceptibility, robustness, capacity and security constraints, issues not discussed within this work..

2.1 Interfaces

It is obvious that watermarking functionality must at least encompass embedding a watermark message into a cover medium and retrieval of a watermark message from a watermarked medium. In addition, both parts of watermarking might require additional parameters special to individual algorithms.

Some parameters like the watermark strength are common to some (or even most) algorithms, some are unique to individual algorithms. This means that there is no chance of standardizing which parameters there need to be mandantorily specified. there existence. Parameters in the AlgorithmManager are thus simply modeled as key-value-pairs.

With respect to the cover, one can differentiate between several ways of representing media: as a file, as a http-, ftp-or rtsp-URL, as a device name (e.g. /dev/audio) or as a stream (as for example done in peer-to-peer networks like Gnutella). This is modeled by the AlgorithmManager as specialized Interfaces for each way of how media is represented.

2.2 WatermarkMessage

The watermark message can be represented in different ways, too. Even though every information can be represented by a binary code, it is often feasible to represent the watermark message as a string of characters or hex values. An embedded ASCII text string whose meaning can be directly understood by any person might be more convincing (e.g. in court) than a binary sequence that has to be interpreted first before it can be understood. Therefore the WatermarkMessage encapsulates both type of representations. Since especially in robust watermarking capacity is a crucial and sparse resource, the binary representation must be bit-exact. But binary sequences are not represented as a sequence of bits; actually they are represented by bytes. Therefore the length of the message is a necessary parameter, without this information 5 bit messages could not be differentiated from 7 bit messages. The same considerations are also valid for the secret key and this class is thus modeled analogously.

2.3 Configuring Algorithms

The previous sections modeled watermarking functionality in a highly abstract way. At some point, a mapping to the concrete algorithms is necessary. This is the task of a configuration file the algorithm developer needs to provide. This (XML-) configuration file provides a mapping from the parameters defined in the embedder and detector interfaces to the parameters understood by algorithm thus building the bridge between algorithm and framework. In addition, it specifies what (other) parameters the algorithm has and what structure these have.

Consider the following example:

```
<algorithm name="MyAlgorithmName">
        <description>
                This algorithm is used for watermarking images with numbers
        </description>
        <mediatypes>
                <mediatype>image/*</mediatype>
        </mediatypes>
        <allowed-characters>[0-9]*</allowed-characters>
        <embedder>
                <embedder-class>
                    mypackage.EmbedderClass
                </embedder-class>
                <ordered-params>
                    <ordered-param name="Param1"/>
                    <ordered-param name="Param2"/>
                </ordered-params>
        </embedder>
        <detector>
                <detector-class>
                    mypackage.DetectorClass
                </detector-class>
                <ordered-params>
                    <ordered-param name="Param2"/>
                </ordered-params>
        </detector>
        <params>
                <param name="Param1" default="true" description="some boolean"/>
                <param name="Param2" default="123456"
                    description="a common numerical parameter"/>
        </params>
</algorithm>
```

This (simple) description contains one algorithm for watermarking images. In addition to the media types that an algorithm is able handle and a regular expression defining the structure of possible watermark messages (here numbers only), the file has three major sections: one describing the embedder, one describing the detector and one defining the parameters. The <embedder-class> tag gives the fully qualified class name of the code that implements the embedder part of the algorithm (detection works analogously). The AlgorithmManager uses a ClassLoader to load this class during runtime . Which parameters the embedding uses (and in what order) is defined in the <ordered-params> section. Since some parameters are common to embedding and detection, they are collectively defined in the <params> section. During runtime, parameters can be set by specifying name-value-pairs. This allows simple configuration of whatever parameters the algorithm requires.

3. Integration

After specifying the integral components of the framework in the previous section, this section will show how algorithms can be integrated into the framework. Please note, that all examples involve embedder interfaces - detection works analogously.

3.1 Java-based watermarking algorithms

Since the framework is written in Java, integration of Java-based algorithms is trivial. As described in section 2.3, the configuration file contains the fully qualified class name of the class implementing the embedder interface(s) thus defining the functionality simply means implementing the interfaces. With the help of the Java Reflection API (java.lang.reflect) the class is then loaded and instantiated during runtime. This method also allows to add or change watermarking algorithms during the runtime of the application thus facilitating maintenance.

3.2 Algorithms provided as C-based Libraries

For other algorithms defined in different programming languages, integration is a bit more complicated. For directly integrating C-based code (i.e. C or C++) into Java the Java Native Interface (JNI) can be used [5]. The keyword native before a Java method signals that the method is externally implemented by native code -like code generated by C compilers. In order to facilitate creation of such code, it is possible to (automatically) create C header files containing the signatures of the methods that have to be implemented. JNI also translates data types and major classes (like Java:String into C++:jstring). So on the Java side, we have implemented a class (called DllEmbedder) that implements embedder and detector interfaces and translates these into ”native” methods. Therefore all algorithm developers implementing in C have to do in order to integrate their algorithms into the AlgorithmManager is to implement this header file and fill out the configuration file specifying the DllEmbedder as the <embedder-class>. This method should also work with libraries written in other programming languages, but this has not been tested yet.

3.3 Algorithms provided as executables

If algorithms are not available as libraries, they are at least available as executable files. If they can be executed in a command line mode (without requiring human interaction) it is also possible to integrate them into the AlgorithmManager framework. This is the task of the ExeEmbedder. This Java class implements the embedder interfaces and uses the information in the configuration file to generate a command string, which is then executed on the command line (using a java.lang.Runtime instance). One challenge of this approach is the transfer of information: Only textual parameters can be included in a command string. Therefore the ExeEmbedder only implements the URIWatermarkEmbedder Interface and watermark messages, which can be binary, are given as files. In a command string identification of parameters is done in two ways: By order or by prefixes (like key ”topSecret”). All this information has to be included into the configuration file, which makes the file a little more complex. The example below shows a watermarking executable that embeds textual messages into MPEG video files. A command line call of this executable might look like this:

```
.\VideoWM_DEW.exe EMBED myCovermedium.mpq myTargetMedium.mpg messageFile.txt topSecret
```

Please note, that for this type of integration it is very hard to exchange information between the application running the AlgorithmManager and the watermark executable (in both directions). Therefore the ExeDetector requires algorithms to output the retrieved message into a file. This spares the ExeDetector from parsing the command line output for the message (see param OUTFILE URI). With the approach of the ExeEmbedder it should be possible to integrate most algorithms into the AlgorithmManager even though the first two approaches are preferable, since they offer more flexibility and simple exchange of information.

4. Status quo, Future Work and Conclusions

Our proposed AlgorithmManager framework is already integrated in our own “Watermarking Portal” described in [4], a generic webservice where users can upload digital media files to embed or detect watermarks (for demonstrational purpose only). The AlgorithmManager is also one important element of our MediaSearch Framework that provides an automated search service for copyright infringements of watermarked protected media files on the Internet.

Next steps in the development of the AlgorithmManager could be the exploration of native approaches that are not based on C. An integration of algorithms implemented with MatLab might proof beneficial. So far the media are simply represented as streams or URIs. This provides only access to the essence (the pure media data). Further information about the media like its mime-type or codec could facilitate selecting of appropriate algorithms. But most importantly, this framework should be tested in many fundamentally different application areas and with many different algorithms to verify if all assumptions prove valid. In addition, the use of Java as an implementation language has to be justified and its influence on critical issues like performance has to be monitored.

Finally, the AlgorithmManager could be used to facilitate translation of user requirements into concrete algorithm parameters. Non-expert users often lack the knowlegde how to parametrize a watermarking algorithm so that it gives them the aspired level of transparency, robustness and security. The AlgorithmManager could help such users to express their wishes in more generic, maybe even non-technical terms and then negotiates the exact parameters with the concrete algorithm (see [7]).

Summarizing, this paper proposed a light-weight framework based on Java called AlgorithmManager. The AlgorithmManager allows application programmers to easily integrate watermarking functionality into their applications independent of concrete algorithms by a set of generic interfaces. Algorithm developers provide implementations of these interfaces and register their algorithms at the AlgorithmManager defining the specifics of their algorithm in a configuration file. It is possible to register algorithms independently of their implementation language, so that for example also C/C++ based algorithms or simply executables can be used within the AlgorithmManager. Frameworks like the AlgorithmManager should pave the way for watermarking to become a black-box functionality that can be easily integrated into other productive applications, easily updated, upgraded and maintained in the productive live system and might be chance to standardize the usage of watermarking.

From the perspective of a user, simple and fast integration of digital watermarking may even be more desirable than having a standard for digital watermarks as the security concepts based on watermarking are often private activities not shared with the public with respect to detectors, embedded information and retrieval results. Therefore a platform providing an universal interface between applications and watermarking software may increase the usability of watermarking dramatically and enable a larger number of users to apply this technology.

Acknowledgements

The work described in this paper has been partially funded by the *CASED* Center of Advanced Security Research Darmstadt" (www.casedweb.de) supported by the Federal State of Hessen, Germany, through the *LOEWE* programme.

References

[1] ARNOLD, M.; HUANG, Z., FAST *Audio Watermarking: Concepts and Realizations*, In: Delp, E.J., Proceedings of Electronic Imaging, Science and Technology; Security, Steganography, and Watermaking of Multimedia Contents Conference VI: January 2004, San Jose, CA, USA. pp. 105-115, 2004.

[2] INGEMAR J. COX, GWENAEL DOERR, and TEDDY FURON. *Watermarking is not Cryptography.* In *5th International Workshop on digital Watermarking, IWDW 2006*, volume 4283/2006, pages 1-15. Springer Berlin / Heidelberg, 2006.

[3] INGEMAR J. COX and MATT L. MILLER. The first 50 years of electronic watermarking. IEEE Journal of applied Signal Processing, 2:126–32, 2002.

[4] FRAUNHOFER SIT. *The Fraunhofer Watermarking-Portal*, March 2006. http://watermarkingportal.sit.fraunhofer.de/.

[5] SHENG LIANG. The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley Professional, 1999.

[6] FABIEN A. P. PETITCOLAS, ROSS J. ANDERSON, and MARKUS G. KUHN. *Attacks on copyright marking systems.* In David Aucsmith, editor, *Information Hiding, Second International Workshop, IH98*, pages 219–239, Portland, Oregon, USA, April 1998. Springer-Verlag.

[7] MARTIN STEINEBACH and JANA DITTMANN. Universelle Parameterübergabe für digitale Wasserzeichen. In Patrick Horster, editor, Tagunsgband DACH Security Bestandsaufnahme und Perspektiven, pages 131–140. IT-Verlag, 2003.

[8] ADRIAN SEQUEIRA and DEEPA KUNDUR. Communication and Information Theory in Watermarking: A Survey. In Proceedings of SPIE, Multimedia Systems and Applications IV, pages 216–227.

[9] MARTIN STEINEBACH, ANDREAS LANG, JANA DITTMANN, and FABIEN A. P. PETITCOLAS. Stirmark benchmark: Audio watermarking attacks based on lossy compression. In Proc. SPIE Security Watermarking Multimedia, pages 79–90.

[10] STEINEBACH, HAUER, WOLF, Efficient Watermarking Strategies, in: Automated Production of Cross Media Content for Multi-Channel Distribution, S. 65-71, IEEE, New York, ISBN: 9780769530307.

[11] STEINEBACH, WOLF, On the necessity of finding content before watermark retrieval: Active search strategies for localizing watermarked media on the Internet, in: Multimedia Forensics and Security, Chang-Tsun Li (edt.), Idea Group Publishing, ISBN: 1599048698 , S. 106- 119, 2008.

]