XML Streams Watermarking*

Julien Lafaye and David Gross-Amblard

Laboratoire CEDRIC (EA 1395) – CC 432
Conservatoire national des arts et métiers
292 rue Saint Martin
75141 PARIS Cedex 3, France
{julien.lafaye, david.gross-amblard}@cnam.fr

Abstract. XML streams are valuable, continuous, high-throughput sources of information whose owners must be protected against illegal redistributions. Watermarking is a known technique for hiding copyrights marks within documents, thus preventing redistributions. Here, we introduce a watermarking algorithm for XML streams so that (i) the watermark embedding and detection processes are done online and use only a constant memory, (ii) the stream distortion is controlled, (iii) the type of the stream is preserved and finally (iv) the detection procedure does not require the original stream. We also evaluate, analytically and experimentally, the robustness of the algorithm against watermark removal attempts.

1 Introduction

Streams. Data streams are *high throughput* sequences of *tokens*, potentially *infinite*. They are used in a growing number of applications (see e.g. [3]) and their specificities make them a challenging application [12]. Since XML has become the standard for specifying exchange formats between applications, the focus in this paper is on XML streams. XML streams can be purchased online and processed by distant peers. Data *producers*, e.g. news providers distributing news item in RSS (an XML dialect) format, generate the tokens of the stream which is later on processed by *consumers*. We focus on automatic consumers, i.e. consumers defined by means of a program (or a Web Service). Hence, consumers, as any program, do not accept arbitrary streams, but place restrictions on their *input types*. Streams with invalid types can not be sold to consumers. For XML based systems, types are usually specified through a Document Type Definition (DTD) or an XML Schema. High throughput requirement puts severe constraints on consumers: they must be able to process each token of the stream quickly and cannot buffer an arbitrary number of tokens (bounded memory). For any arbitrary DTD, typechecking XML streams can not be done while respecting these constraints. Hence, we focus on *acyclic* DTDs , where no element is a subelement of itself (for example, RSS is an acyclic DTD). Under this hypothesis, typechecking can be done using deterministic finite automata (DFA) and types can be specified using regular expressions [15].

* Work supported by the ACI Sécurité & Informatique TADORNE grant (2004-2007).

E. Damiani and P. Liu (Eds.): Data and Applications Security 2006, LNCS 4127, pp. 74–88, 2006.

Example 1. The Xml news feed of Fig. 1 may be regarded as a stream on an alphabet of *closing and ending tags* (< news >, < /date >..), *letters* (S,o,d,e,1,...) and predefined *sequences of letters* (Cinema, Politics, ...). It can be typechecked using the regular language `<news><priority>[123]</priority> <title>(.*)</title>..<date>`*D*`</date>...</news>`, where the expression `D=(19|20)[0-9][0-9]-(0[1-9]|1[0-2])-(3[0-1]|0[1-9]|[1-2][0-9])` captures valid dates (for simplicity we do not try to check dates like `2005-02-31`). Observe that the DTD standard does not allow the definition of a precise date format, since the contents of elements are mostly of type PCDATA (i.e. almost any sequence of letters). A more sophisticated model like Xml Schema allows for such precise definitions. Our model applies to both formalisms.

```
...</news><news>
  <priority>1</priority>
  <title>Soderbergh won the Golden Palm</title>
  <url>http://www.imdb.com/title/tt0098724/</url>
  <date>1989-05-23</date>
  <text>Soderbergh's movie, Sex, lies and videotapes, won the ...</text>
  <category>Cinema</category>
</news><news>...
```

Fig. 1. An Xml stream snapshot

Watermarking. High-quality streams carry a great intellectual and/or industrial value. Malicious users may be tempted to make quick profit by stealing and redistributing streams illegally. Therefore, data producers are interested in having a way to prove their ownership over these illicit copies. *Watermarking* is known to bring a solution to that issue by hiding copyright marks within documents, in an imperceptible and robust manner. It consists of a *voluntary alteration* of the content of the document. This alteration is parameterized by a key, kept secret by the owner. Accordingly, the secret key is needed to detect the mark and thus, to prove ownership. The robustness of the method relies on the key in the sense that removing the mark without its knowledge is very difficult. A first challenge of streams watermarking is to control and minimize the alteration of the stream, i.e. to *preserve* its *quality*. We measure the alteration by means of a relative edit-distance and propose a watermarking algorithm that introduces a bounded distortion according to this measure. A second challenge is to *preserve the type of the stream* so that it remains usable by its intended consumers. Existing Xml watermarking schemes embed watermarks by modifications of the content of text nodes. We believe that other embedding areas may be used, e.g. within the tree-like structure itself. Obviously, altering the structure can not be done naïvely. For instance, in some pure text watermarking schemes, bits are embedded by switching words of the document with their synonyms. This can not be directly applied to our context: if the name of an opening tag is switched, the corresponding closing tag has to be switched to ensure well-formedness. Even

if tag names are switched consistently, the resulting document may become invalid with respect to its original type. In that case, watermarked documents are *unusable* by their target consumers. Remark also that a good watermarking method must be *robust*, i.e. still detects marks within streams altered (random noise, statistical analysis, ..) by an attacker (up to a reasonable limit).

Our Contribution. In this paper, we introduce the ℓ-`détour` algorithm, a robust watermarking scheme for XML streams, which respects the quality of the stream as well as its type, specified by means of an acyclic DTD. The idea of ℓ-`détour` is the following. We identify two relevant parts of the stream, based on its semantics. The first *unalterable* part can not be altered by any attack without destroying the semantics of the stream. The second *alterable* part is still useful for the application, but can be altered within reasonable limits. For the automaton of Figure 1, the *unalterable* part will be e.g. the path name in the `url` element (but not the host name, since it can easily be replaced by an IP number). The alterable part will be e.g. the two digits of the day in the `date` element. Alterable parts can capture purely textual information as well as structuring one. A finite portion of the *unalterable* part, combined with a secret key known only by the data owner, is used to form a *synchronization key*. A non-invertible (cryptographic) pseudo-random number generator, seeded with this synchronization key, determines how the *alterable* part of the stream is modified to embed the watermark. This process, repeated along the stream, introduces *local dependencies* between parts of the data stream. These dependencies, invisible to anybody who does not possess the key used for watermarking, are checked at detection time by the owner. Only the private key and the suspect stream are needed. It can be viewed as an extension of Agrawal and Kiernan's method [2] which considered relational databases watermarking (primary keys played the role of our synchronization keys). In order to respect the type constraint, we simulate the DFA that typechecks the stream. Each time the insertion of a dependency is required, we change a sequence of tokens of the stream so the walk on the automaton follows a *detour*, leading to the *same* state. If the altered sequence lead to state q, the chosen detour still leads to q. The length ℓ of the detours and the frequency of the alteration control the quality of the stream. The DFA is also used to define the alterable and unalterable parts of the stream.

Organization. In Section 2, we present our main contribution: the ℓ-`détour` algorithm, which allows for watermarking XML streams so that (i) the watermark embedding and detection processes are done online and use only a constant memory, (ii) the stream distortion is controlled, (iii) the type of the stream is preserved and finally (iv) the detection procedure does not require the original stream. In Section 3, we discuss on the robustness of ℓ-`détour` against attempts to remove the watermark and show that attackers have to alter more the streams than the watermarking process did to remove the mark. Comparison with related work is presented in Section 4. Section 5 concludes.

2 The ℓ-détour Algorithm

2.1 Preliminaries

In this paper, we use ω-rational languages on words, i.e. a simple, yet expressive, extension of regular languages suited to infinite words.

- **Streams:** *Let Σ be a finite alphabet. Letters from Σ are called tokens. A Σ-stream σ is an infinite sequence of tokens from Σ.*
- **Stream Automaton:** *A stream automaton is a deterministic finite state automaton such that all states are accepting, except one which has no outcoming edge to an accepting state. This state is called the blocking state.*
- **Stream Acceptance:** *Let G be a stream automaton. A stream σ is accepted by G if the walk on G due to σ never enters the blocking state.*
- **Stream Types:** *A set of streams $\mathcal{L}$ is a stream type if there exists a stream automaton G such that $\mathcal{L}$ is the set of all streams accepted by G.*

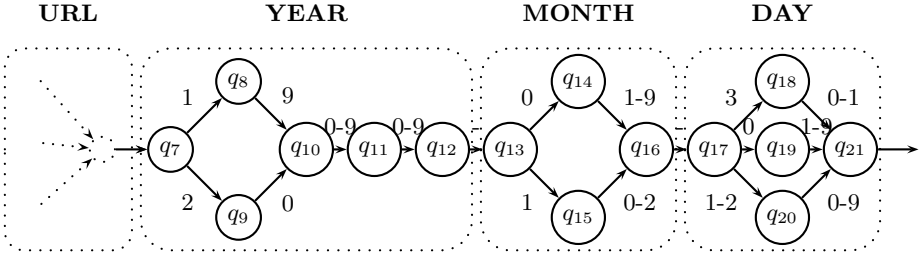

Fig. 2. A partial specification of the stream type for news items (`date` element)

Example 2. Figure 2 shows a partial specification of a stream automaton for the input type of a news items consumer. It checks that the syntax of the date is correct. The part checking that the stream is well-formed and conforms to the complete DTD is not depicted here. All unspecified transitions lead to the blocking state.

As a means to measure the distortion introduced by watermarking algorithms, we introduce the *relative edit-distance*. It is based on the edit-distance for strings [10]. In our context, the edit-distance $d_e(x, y)$ between words x and y is defined as the minimum number of operations (substitution/deletion/insertion of a token) that are needed to transform x into y. For instance, if y has been obtained by substituting one symbol of x, $d_e(x, y) = 1$. The *relative edit-distance* between x and y is defined as the average number of operations per symbol that are needed to transform x into y. We measure the *relative edit-distance* from finite prefixes of streams:

Definition 1 (Distance). *Given σ^N (resp. σ'^M) a finite initial segment of a stream of length N (resp. M), the relative edit distance $d(\sigma^N, \sigma'^M)$ is defined by:*

$$d(\sigma^N, \sigma'^M) = \frac{d_e(\sigma^N, \sigma'^M)}{\sqrt{N}\sqrt{M}}.$$

Example 3. $d(babba, dabba) = 1/5$. Letter b has been substituted for d (edit-distance 1), and both words have length 5.

2.2 Informal Introduction to ℓ-détour

Suppose that we want to watermark a data stream σ flowing from a producer $\mathcal{P}$ to a consumer $\mathcal{C}$ whose input type is specified by a stream automaton G. Since $\mathcal{P}$ produces a usable stream for $\mathcal{C}$, its outputs correspond to non blocking walks on G. Assume that there exist in G two different edges (paths of length 1), labelled by different tokens, and having same start and same end (for example, paths from q_{17} to q_{20} in Fig. 2). These edges can be loops on a single node. The idea of our algorithm is to change the value of some tokens of the stream so that the walk on G follows one of these edges rather than the other (for instance, $q_{17} \xrightarrow{1} q_{20}$ instead of $q_{17} \xrightarrow{2} q_{20}$). These tokens are chosen as a function of (1) the secret key K_p of the owner and (2) a finite portion, carefully chosen, of the path previously covered. The original walk on the automaton is diverted, and becomes specific to the data owner. This process is repeated along the stream. Notice that following an edge once does not imply that it will always be chosen because the path previously covered varies. Then, a watermarked stream is composed of alternated sequences of *unaltered* segments (*synchronization* segments) and *altered* segments of length 1. The value of an altered segment cryptographically depends on the value of its preceding synchronization segment. This method ensures that the type of the stream is respected. Furthermore, the modified stream is close to the original: each choice between two different paths adds at most 1 to the *edit-distance* between the original and the watermarked stream (and less to the relative edit-distance).

2.3 Finding Detours

The previous paragraph gave the idea of the 1-détour algorithm because paths of length 1 were altered in order to embed the watermark. The extension of this algorithm to path of length exactly ℓ is given the name of ℓ-détour. In ℓ-détour, not all paths of length ℓ may be changed but only those called *detours*:

Definition 2 (Detours). *Let G be a stream automaton. The path $p = q_i \to ... \to q_j$ is a detour of length ℓ in G if its length is ℓ and if there is no path p' in G, distinct from p, of length at most ℓ, having the same end points q_i and q_j, and an internal node in common.*

Example 4. In any stream automaton, all edges are detours of length 1 since they do not contain any internal node. Remark also that as soon as $\ell > 1$, cycles are not allowed in detours of length ℓ. On the automaton of Fig. 2, there are detours of length 2: $q_7 \xrightarrow{1} q_8 \xrightarrow{9} q_{10}$ and $q_7 \xrightarrow{2} q_9 \xrightarrow{0} q_{10}$. Conversely, paths from q_{13} to q_{16} going through q_{14} are not detours because q_{14} is an internal node common to 9 paths of length 2 between q_{13} and q_{16}. There are 9 paths from q_{14} to q_{16} labeled by 1 to 9.

The proof of proposition 1 provides a constructive way to compute detours. Due to space reasons, it is not detailed. Remark that space complexity of the method is $O(n^2|\Sigma|\ell)$ whereas it is usually $O(n^2|\Sigma|^\ell)$ to compute paths (and not detours) of length ℓ.

Proposition 1. *Let Σ be the alphabet, n the number of states of the automaton and $\ell \in \mathbb{N},\ \ell > 0$. Detours of length ℓ can be computed in space complexity $O(n^2|\Sigma|\ell)$ and time complexity $O(n^3\ell)$.*

Proof. (sketch) Since detours are paths, i.e. finite sequences of labelled edges, a first naïve strategy is to compute the set of paths of length ℓ and remove paths which are not detours. If $S^k(i,j)$ is the set of paths of length k between states i and j, the formula $S^{k+1}(i,j) = \bigcup_{q \in states(G)} S^k(i,q) \times S^1(q,j)$ permits to define an iterative algorithm to compute $S^k(i,j)$ for any $k > 0$ (if R, S are two sets, $R \times S$ is defined as the set containing the concatenation of every item of R with every item of S). Unfortunately, this leads to an exponential blowup because the number of paths of length ℓ is $n|\Sigma|^\ell$ in the worst case. This blowup can be avoided by getting rid of paths which will not become detours, at each iteration. Indeed, if p, p' are two detours having the same end points and e is an edge in G, $p.e$ and $p'.e$ are not detours because they share an internal node: $end(p) = end(p')$. This fact remains true for any two paths which have p and p' as prefixes. Similarly, if p is a detour of length k between i and q and e, e' are two edges between q and j, $p.e$ and $p.e'$ are not detours. Hence, we can reduce the number of paths which are detours in the sets computed by the naïve algorithm by modifying the definition of the $\times$ operator: if R and S are not singletons, $R \times S = \emptyset$. This can be checked in constant time. Another condition is necessary to strictly compute sets of detours: if p_1 (resp. p_2) is the only detour of length $k > 1$ between states i and q_1 (resp. q_2) and e_1 (resp. e_2) is the only edge between states q_1 (resp. q_2) and j, $p_1.e_1$ and $p_2.e_2$ are detours of length $k+1$, unless p_1 and p_2 share their first edges. To check this when computing $R \times S$, buffering only the first edge of each path in R is needed. There are at most $|\Sigma|$ such edges.

This leads to a time complexity $O(n^3\ell)$ and a space complexity $O(n^2|\Sigma|\ell)$. At each of the ℓ iterations, there are n^2 sets of detours to compute, each step requiring at most n operations. Space complexity is $O(n^2|\Sigma|\ell)$ because the number of detours is at most $|\Sigma|$ between any two states (two detours can not begin with the same edge). There are n^2 pairs of states and the maximum length of a detour is ℓ.

Interesting detours are likely to be found in real applications. For example, there are 9 detours of length 2 in the Rss specification, 39 detours of length 1 in a valid email addresses recognizer, and 48 detours of length 1 in a checker of valid IP numbers. In the sequel, only detours of length exactly ℓ are used. A straightforward extension not shown here allows for using all detours of length at most ℓ.

2.4 Watermark Embedding

The ℓ-`détour` algorithm can be divided into three successive steps. Steps (1) and (2) are performed once for all, while step (3) is used online and requires constant memory.

(1) *Choice* of the automaton and *Precomputation* of the detours given a target detour length ℓ.
(2) *Annotation of the automaton.* The set of detours is split up into the set of *alterable* ones and the set of *unalterable* ones. Among the set of remaining edges (i.e. edges not part of a detour or part of an unalterable detour), a subset of *synchronization* edges is selected.
(3) *On-the-fly watermarking.* The stream is continuously rewritten by substituting some sequences of ℓ tokens.

STEP 1: *Precomputation.* For a given input type, a canonical choice for the stream automaton is the minimal deterministic recognizer of the DTD, but any equivalent deterministic recognizer may be used. A strategy is to start with the minimal one and to compute the detours using Prop. 1. If their number is too small or if they do not fit the owner's needs, the automaton can be unfolded into an equivalent one by splitting nodes and duplicating edges, and detours recomputed.

STEP 2: *Annotation of the automaton.* Not all detours are suitable for watermarking. For instance, on Fig. 2, there are two detours of length 2 between states q_7 and q_{10}: $q_7 \xrightarrow{1} q_8 \xrightarrow{9} q_{10}$ and $q_7 \xrightarrow{2} q_9 \xrightarrow{0} q_{10}$. Using these detours for watermark embedding would imply changing the millennium of a news item, resulting in an important loss of semantics. A solution is to divide the previously computed set of detours into two subsets: the subset of *alterable* detours and the subset of *unalterable* ones. This partition is done by the owner based on semantical criteria. All the remaining edges can not be used as *synchronization edges*. Indeed, some of them may be changed by an attacker without too much altering the semantics of the data which would result in the impossibility to resynchronize during the detection process and makes the watermark ineffective. For instance, we should not use the title as synchronization key because it can be altered, e.g. by adding spaces or changing the case of some characters, without changing its semantics. Conversely, the path in the `url` is not likely to be changed in an uninvertible manner (e.g. replacing letter `'a'` by code `%61`). The corresponding edges in the automaton can be chosen as *synchronization* ones.

Example 5. A natural choice for watermarking news items is to modify the least significant part of the date. This can be achieved by using only detours from states q_{17} to q_{20}, detours from states q_{18} to q_{21} and detours from states q_{19} to q_{21} as *alterable* ones.

STEP 3: *On-the-fly Watermarking.* In this last step, the core of ℓ-`détour`, some portions of the stream are changed to insert the watermark. It is called `streamWatermark` and sketched on Fig. 3. Its execution is basically a walk on the automaton used to typecheck the stream. At each move, the last covered edges are changed if they match an alterable detour of length ℓ. Inputs of `streamWatermark` are a stream σ, the private key K_p of its owner and an extra parameter γ used to change the alteration rate (on average, one alterable detour out of γ is altered).

The `streamWatermark` procedure uses two variables: p and K_s. The path p is a finite queue having size at most ℓ containing the last covered edges, used as a finite FIFO: before adding a new edge at the end of a full p, its first edge is discarded. When p is full, it contains a candidate detour, likely to be changed if it matches an *alterable* detour. The second variable K_s stands for the synchronization key. It is used as a bounded-size queue of tokens. It will contain any symbol that corresponds to a synchronization edge.

The `streamWatermark` algorithm starts in **A** and regularly loops back to this cell. In **A**, we read a token from the input stream which generates a move on the automaton. The covered edge is added to p. Then, we move to cell **B**. If length(p)$< \ell$, we move back to **A**. When length(p)$= \ell$, we move to **C**. In cell **C**, we test whether p is going to be changed i.e. whether p is an alterable detour (from states i to j) and whether there is at least one another other detour from i to j. When these two conditions are met, we move to the watermark cell **E**. In **E**, the path p is converted into an integer: its rank in an arbitrary ordering of all detours from i to j. This integer, together with the synchronization key K_s, the private key of the owner K_p and γ, is passed to the procedure `intWatermark` (Alg. 1). Its output is the number of a new detour whose labelling symbols will be added to the output stream. This procedure, derived from [2], uses a pseudo-random generator $\mathcal{R}$ seeded with $K_\mathrm{s}.K_\mathrm{p}$ to choose (1) whether the passed integer is going to be altered or not (2) which bit of the passed integer is going to be modified and (3) what will the new value of this bit. The synchronization key K_s is reseted to the empty queue. Remark that this modification only depends on the private key of the owner and tokens of the stream which are not altered. If the conditions to move to cell **E** are not met, we move to cell **D**. Path p not being an alterable detour does not mean that its suffix of length $\ell - 1$ is not the prefix of another detour. So, in **D**, the first edge of p is discarded and, if it is a synchronization edge, its labelling token c added to K_s. Simultaneously, c is added to the output stream. The process loops back to the initial cell **A**.

Hence, the ℓ-`détour` algorithm outputs 0,1 or ℓ tokens every time it reads a token from the input stream. If N tokens have been read from the input stream, at least $N - \ell$ and at most N tokens have been outputted which makes the process a real-time one. The output of `streamWatermark` is a stream of the form

$c_1e_1c_2e_2...$ where each c_i comes from the input stream and e_i is the result of a pseudo-random choice seeded with the synchronization part of c_i concatenated with the private key of the owner. Each segment e_i has length ℓ.

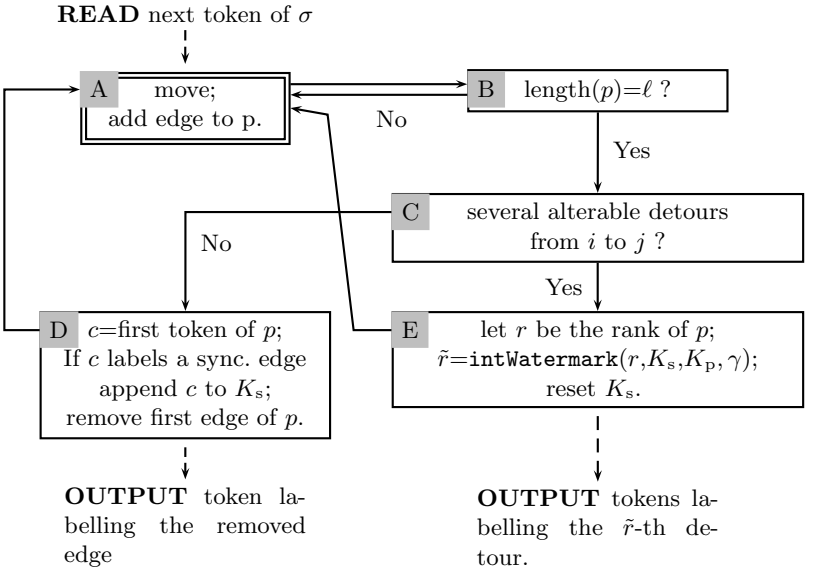

Fig. 3. `streamWatermark`$(\sigma, K_\mathrm{p}, \gamma)$

Example 6. Suppose that we are in the middle of the watermarking process of the XML segment of Fig. 1. Detours of length $\ell = 1$ have been chosen and the partition of detours has been done in Example 5. Suppose also that the algorithm has just reached cell **A**, that the current position on the automaton is state q_{13} (last read token is `-`), that $K_\mathrm{s} = K_\mathrm{s}^0 =$`<url>http://www.imdb...</url>` and $p = q_{12} \xrightarrow{\text{-}} q_{13}$. The path $q_{12} \xrightarrow{\text{-}} q_{13}$ has length 1 but is not a detour, so we move to cell **D** through cell **C**. In cell **D**, the first token of p, `-` is removed, appended to K_s and added to the output stream. Then, $p = []$ and we move to cell **A**. The token `0` is read from the input stream and the edge $q_{13} \xrightarrow{0} q_{14}$ appended to p. Still, p is not an *alterable* detour and the same sequence of steps through cells **B,C,D** is performed. Then, the algorithm moves through edges $q_{14} \xrightarrow{5} q_{16}$, $q_{16} \xrightarrow{\text{-}} q_{17}$ and $q_{17} \xrightarrow{2} q_{20}$; the tokens `5,-,2` are processed the same way the token `0` was. The token `3` coding for the lowest significant digit of the day in the month is read in cell **A**. The path $p = q_{20} \xrightarrow{3} q_{21}$ is a detour of length 1 from states q_{20} to q_{21}. Since there are 10 detours between these states, we move to watermarking cell E. The `intWatermark` procedure is called with $K_\mathrm{s} = K_\mathrm{s}^0$`.05-2` and $r = 4$ (p is the fourth detour from q_{20} to q_{21}). A one-way cryptographic choice of a new detour is done by Alg. 1, depending only on K_s and K_p. For instance, if `intWatermark` outputs 7, the seventh detour is chosen and the token `6` added to

Algorithm 1. `intWatermark`(i,K_s,K_p,γ)

Output: $1 \leq j \leq n$
$\mathcal{R}$.seed(K_s.K_p); `/* seed the random generator */`
`// (1) decide whether` i `is going to be changed`
if $\mathcal{R}$.*nextInt()* % γ = *0* **then**
  $p = \mathcal{R}$.nextInt() % $\lceil \log_2(n) \rceil$; `/* (2) choose which bit of` i `to change */`
  $b = \mathcal{R}$.nextInt() % 2; `/* (3) new value of bit` p `of` i `*/`
  $j := i$ where bit p is forced to b;
  `return` j;

the output stream. The watermarked date is `1989-05-26`. Then, K_s and p are reseted and we loop back to **A**.

2.5 Quality Preservation: Setting Alteration Frequency γ

The following theorem quantifies to what extent the quality of a watermarked stream is preserved. Let G be a stream automaton. Let S (resp. E) be the set of starting (resp. ending) nodes of the *alterable* detours. We define the *inter-detours* distance c as the length of the shortest path between a node in $E \cup q_0$ and a node in S. For the automaton of Fig. 2, $\{q_{17}, q_{18}, q_{19}\} \subseteq S$ and $\{q_{20}, q_{21}\} \subseteq E$ so c is at most the minimum of the distances between q_0 and q_{17} and between q_{21} and q_{17} (the actual *inter-detours* distance can not be given because of the partial specification).

Theorem 1. *Let σ^N a finite prefix of a stream and $\tilde{\sigma}^N$ its watermarked version using ℓ-`détour`. Then, at most $d(\sigma^N, \tilde{\sigma}^N) \leq (1 + \frac{c}{\ell})^{-1}$ and on average $d(\sigma^N, \tilde{\sigma}^N) \leq \frac{1}{\gamma}(1 + \frac{c}{\ell})^{-1}$.*

Proof. A finite segment σ^N of a stream σ can be written as $\sigma^N = c_1 e_1 .. c_n e_n r$ where $c_1, .., c_n$ are token sequences used as synchronization keys, $e_1, .., e_n$ are token sequences labelling detours and r is the remaining. ℓ-`détour` introduces a distortion of at most $n\ell$. Since the length of each c_i is at least c, the relative distortion $\varepsilon = \frac{n\ell}{\sum_{1,n} |c_i| + n\ell + |r|}$ is such that $\varepsilon \leq (1 + c/l)^{-1}$. On average, $\frac{1}{\gamma}$ pairs $c_i e_i$ are altered.

Hence, for a maximum error rate $e = 0.1\%$, a detour length $\ell = 2$ and an inter-detour distance $c = 10$, the value of γ is chosen so that $\frac{1}{\gamma}(1 + \frac{c}{\ell})^{-1} \leq e$ i.e. $\gamma \approx 6000$. So, on average, one over 6000 tokens labelling *alterable* detours should be altered to comply with this error rate.

2.6 Watermark Detection

Since the alterations performed by the watermarking process depend only on the value of the private key K_p of the owner, exhibiting a key and making the dependencies appear is a strong proof of ownership. The detection process locates

the synchronization keys and checks whether the detours taken by the suspect stream match what would be their watermarked value. It is very close from the watermarking algorithm except that the content f the stream is not changed. We use two counters, tc and mc, tc standing for *total count* and mc for *match count*. We increment tc every time we meet a detour that would be watermarked (this corresponds to line 2 of Alg. 1). We increment mc every time a detour matches what would be its watermarked value. Therefore, $tc \geq mc$. When $tc = mc$, we can conclude of the presence of a watermark. When $tc > 0, mc = 0$, we are probably in front of an attacker who successfully inverted every bit of the mark. This inversion is considered as suspicious as the full presence of the mark (think of it as a negative image of a black and white picture). For a non-watermarked stream, we can assume that there is no correlation between the distribution of the data and the pseudo-random watermark embedding process (assumption verified in our experiments). In this case, the probability that each bit of a detour matches what would be its watermarked value is $1/2$. Then, we can await for tc to be twice the value of mc when there is no mark. To sum up, the watermark is found when $|mc/tc - 1/2| > \alpha$, where α is a predefined threshold. The choice of α is very important: if α is too large, the detection raises false alarms; if α is too small, slightly altered marks become undetectable, raising false negatives. The choice of α is discussed in the next section. Remark also that only the suspect stream and the private key of the owner are needed to check for a watermark.

3 Robustness: Analysis and Experiments

A watermarking algorithm is said to be *robust* when an attacker, unaware of the secret key used for watermark embedding, has to alter more the data than the watermarking process did, in order to remove the mark. In that case, the attacked stream suffers a huge loss of semantics, which is very likely to destroy their quality.

3.1 Synchronization Attacks

A watermarked stream can be attacked by modifying synchronization parts. Indeed, ℓ-`détour` requires these parts to remain identical for detection. Such attacks are limited by the *constant requirement to keep streams valid with respect to the input type of their consumers*. A non-valid stream cannot be resold by a malicious user. As explained in **STEP 2** of ℓ-`détour`, synchronization parts are chosen to be semantically relevant which means that they cannot be changed without widely affecting its semantics. Therefore, a type breaking attack requires to alter data semantics more than the watermarking process did.

3.2 Detours Attacks

Since the attacker is unaware of which detours were actually altered, two strategies are available to him. First, he can try to remove the mark by randomly modifying the altered detours. We model this attack as a *random attack*.

Random Attack. For $0 < p < 1$, a random attack of parameter p is an attack inverting each bit of the watermark with a probability at most p. The false negative occurrence probability $p_{fn}(p)$ is the probability that an attacker performing a random attack of parameter p cheats the detector. Theorem 2 (see [7] for a complete proof) shows how to choose α (detection threshold) and tc (number of altered detours to poll) to get this probability maximally bounded by an owner-defined probability δ (e.g. $\delta = 10^{-6}$). These parameters also allows for a false positive occurrence probability p_{fp} bounded by δ.

Theorem 2. *Let* $0 < \delta, p < 1$, $tc \in \mathbb{N}$, $p \neq 1/2$, $tc_0 = \frac{-\log(\delta/2)}{2(1-2p)^2}$, $\alpha_1(tc) = \frac{-\log(\delta/2)}{2tc}$ *and* $\alpha_2(tc) = \frac{1}{2} - p - \sqrt{\frac{-\log(\delta/2)}{2tc}}$. *Then,*

$$(tc \geq tc_0 \text{ and } \alpha_1(tc) \leq \alpha \leq \alpha_2(tc)) \Rightarrow (p_{fp} \leq \delta \text{ and } p_{fn}(p) \leq \delta).$$

Proof. (sketch) The fact that a detour matches its watermarked value is seen as the outcome of a Bernoulli's law of parameter $1/2$. Suppose that we are able to retrieve n possibly watermarked positions in the stream. The probability that a false positive occurs is exactly the probability that the number of positive outcomes in n outcomes of Bernoulli's experiments deviates from the standard value $n/2$ by a distance $\alpha.n$. The higher n is, the smaller this probability. It can be bounded using a Hoeffding inequality [8] to obtain a maximal bound for the occurrence of a false positive. Similarly, one can bound the probability that a false negative occurs. By combining these two results, we find the minimum number of potentially watermarked detours one must consider to test the presence of a watermark and simultaneously stay under the target probability δ.

Listen & Learn Attack. When a synchronization key is met twice, the two corresponding watermarked bits will have the same position and the same value. If $c.e_1$ and $c.e_2$ are two sequences *synchronization key.detour*, the watermarked bit is among the set of bits which have the same value in the binary representations of the rank of e_1 and the rank of e_2. An attacker may try to learn such dependencies in order to perform a *Listen & learn* attack. This attack consists of the following two steps. First, *learning* associations between synchronization keys values and watermarked bits. Second, *attacking* the watermark using this knowledge. Notice that this can not be done in constant memory and requires an external and efficient storage. This does not comply with computational constraints on streams, that also apply to the attacker.

3.3 Experiments

Test Sample. We used RSS news feeds provided by CNN [1] from September 8th 2005 to September 14th 2005 for a total of 1694 news items (or 523041 tokens). *Alterable* detours were chosen to be the edges associated to the highest significant digit of the minutes field and the lowest significant digit of the seconds field. Hence, dates of news item are changed by at most 50 minutes and 9 seconds. Synchronization keys include the content of the `link` element and edges not part of an alterable detour in the `pubDate` element.

Detour-witching Attack. A *detour-switching* attack consists of randomly switching *all* alterable detours. It is parameterized by the alteration frequency q: with probability $1/q$ each detour is replaced by another one, having same start and same end, randomly chosen. We performed experiments for various values of q and γ. A summary of the results is displayed in Table 1(a). For each combination of q and γ, the set of news items was watermarked and attacked 100 times. We count the number of positive detections **PD** of the watermark and the relative extra alteration **QL** introduced by the attack, compared to the watermarking process. If the watermarking process alters **WL** tokens of the stream, then the attack has an overall distortion of **WL**+**QL**. For instance, when $q = 1$ and $\gamma = 3$, the attack successfully erases the watermark (**PD**= 0%) but at the price of a significant quality loss **QL**= 0.39% compared to the alterations introduced by the watermarking process **WL**= 0.22%. On the contrary, for $q = 1$ and $\gamma = 1$, the attack is a success (**PD**= 0%, **QL**= 0%). This shows that choosing $\gamma = 1$ is a bad idea for the data owner, as it means watermarking *every* possible position, hence giving a severe hint to the attacker. As soon as $\gamma > 1$, the mark is not removed if the attack does not alter more the stream than the watermarking process did.

Table 1. Attack Experiments: **WL** (quality loss due to watermarking), **QL** (*extra* quality loss due to attacks), **PD** (ratio of positive detections)

q \ γ	1 (high rate)	2	3 (low rate)
1	**WL**:0.65% **QL**:0% **PD**:0%	**WL**:0.32% **QL**:0.38% **PD**:0%	**WL**:0.22% **QL**:0.39% **PD**:0%
2	**WL**:0.65% **QL**:0% **PD**:100%	**WL**:0.32% **QL**:0.19% **PD**:100%	**WL**:0.22% **QL**:0.19% **PD**:100%
3	**WL**:0.65% **QL**:0% **PD**:100%	**WL**:0.32% **QL**:0.13% **PD**:100%	**WL**:0.22% **QL**:0.13% **PD**:100%

(a) Random Attack
Failure probability $\delta = 0.01$

strategy \ *ltime*	100	500	1500
surge	**QL**:0.27% **PD**:100%	**QL**:0.39% **PD**:100%	**QL**:0.24% **PD**:100%
destructive	**QL**:0.57% **PD**:52%	**QL**:0.49% **PD**:100%	**QL**:0.27% **PD**:100%

(b) Listen & Learn Attack
$\delta = 0.01$,$\gamma = 3$ and **WL**= 0.22%

Listen & Learn Attack. We performed experiments of this attack using two strategies. In the *destructive* strategy we change every *alterable* detour unless we know it is a watermarked one. In the *surge* strategy, a detour is altered only if we are sure it is a watermarked one. We performed experiments for different learning times, *ltime* ranging from 100 detours to 1500. The detection process begins after the end of the learning period to maximize the effect of the learning attack. For each strategy and learning time combination, 100 experiments were performed. Results are presented in Table 1(b). In only one case, the watermark is removed. This is not surprising because when $ltime = 100$, the destructive strategy is a random attack with $p = 1$. Indeed, not enough knowledge has been acquired. Even for longer learning times, the attack does not affect the detection.

4 Related Work

Our work is an extension of [2] which considered relational database watermarking. In [2] and its further extension [11], the watermarked information is located in the least significant bits of numerical values whereas ours is located at any position, provided this position can be localized by an automaton. Type-preservation is implicit since the structure of the databases (relation name, attribute names, key constraints) is not altered. In the XML context, structure is far more flexible and can be used to embed watermarking bits. This motivates structural modifications in the purpose of watermarking, but while keeping the data usable, i.e. respecting its original type. Such structural modifications are not discussed in [2]. It is noteworthy that our automata-based model can mimic their algorithm for numerical values with a fixed size (which is a usual hypothesis in practice).

In [16], a watermarking scheme for sensor streams is proposed. Streams are defined as continuous sequences of numerical values. Watermarking is based on a continuity hypothesis and is performed by altering salient points of the stream. This method does not consider typing problems. Keys and alterations are to be found in numerical values, whereas this can change in our approach according to the form of the stream.

Other works [6,4,13,17,9,14] address watermarking XML information in various contexts. In all these works XML documents are viewed as a whole, and not as streaming information. In [4,17,9,13], watermark embedding values are located through the use of specific XPATH queries. It is not discussed whether these techniques can be applied in a streaming context but it must be observed that XPATH can not be efficiently evaluated over streaming data [5]. Only one work [9] considers structural modification as bandwidth for watermarking which are often viewed as attacks [17,14] watermarkers must deal with. A theoretical work [6] explores the watermarking of XML databases while preserving constraints which are specified trough parametric queries. Type and stream constraints does not fit this framework.

5 Conclusion and Future Work

In this work, we have presented the ℓ-`détour` algorithm which permits the embedding and the detection of copyright marks into XML streams. Thus, it enables detections of illegal redistributions of such objects. Future work is to study whether it is possible to detect watermarks after one or several transformations by consumers. Obviously, this is impossible in the most general setting but preliminary results [7] show that this question can be answered for a restricted class of transformations, expressing deterministically invertible stream rewritings.

References

1. CNN RSS `http://www.cnn.com/services/rss/`.
2. R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking Relational Data: Framework, Algorithms and Analysis. *VLDB J.*, 12(2):157–169, 2003.

3. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.
4. D. G.-A. Camelia Constantin and M. Guerrouani. Watermill: an optimized fingerprinting tool for highly constrained data. In *ACM Workshop on Multimedia and Security (MMSec)*, pages 143–155, August 1-2 2005.
5. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.
6. D. Gross-Amblard. Query-preserving watermarking of relational databases and XML documents. In *Symposium on Principles of Database Systems*, pages 191–201. ACM, 2003.
7. D. Gross-Amblard and J. Lafaye. Xml streams watermarking. Technical report, CEDRIC, 2005. CEDRIC TR-976.
8. W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.
9. S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto, and H. Nakagawa. Proposal on information hiding method using xml. In *The 1st Workshop on NLP and XML*, 2001.
10. V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10(7):707–710, 1966.
11. Y. Li, V. Swarup, and S. Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE Trans. Dependable Sec. Comput.*, 2(1):34–45, 2005.
12. B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A Transducer-Based XML Query Processor. In *VLDB*, pages 227–238, 2002.
13. W. Ng and H. L. Lau. Effective approaches for watermarking xml data. In *DASFAA*, pages 68–80, 2005.
14. M. A. Radu Sion and S. Prabhakar. Resilient information hiding for abstract semistructures. In S. Verlag, editor, *Proceedings of the Workshop on Digital Watermarking IWDW*, volume 2939, pages 141–153, 2003.
15. L. Segoufin and V. Vianu. Validating Streaming XML Documents. In *Symposium on Principles of Database Systems*, pages 53–64, 2002.
16. R. Sion, M. Atallah, and S. Prabhakar. Resilient Rights Protection for Sensor Streams. In *Proc. of the 30th International Conference on Very Large Data Bases*, Toronto, 2004.
17. X. Zhou, H. Pang, K.-L. Tan, and D. Mangla. Wmxml: A system for watermarking xml data. In *VLDB*, pages 1318–1321, 2005.