Forensic Science International: Digital Investigation 58 (2026) 302147

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

Reasoning about artefact tampering

Christopher Neale [a,*], Ian Kennedy [a], Bashar Nuseibeh [b]

[a] The Open University, United Kingdom
[b] City St George's, University of London, United Kingdom

ARTICLE INFO

Keywords:
Tampering
Anti-forensics
Digital artefacts
Modelling
Temporal logic

ABSTRACT

Digital forensic investigations can be disrupted when actors tamper with digital artefacts on devices before they are seized. This activity, when undetected, misleads investigations and results in incorrect findings. Several established research approaches in digital forensics use formal models to identify whether a specific action has occurred through event reconstruction. However, we address two specific concerns which limit their use when investigating tampering actions. First, current understanding of tampering is based solely on informal natural language descriptions; no representations of tampering exist that can be used with the reasoning applied with formal models. Second, mathematically solving these types of reconstruction problems relies on generating relevant input models and analysing their state-space, which is considered to be impractical for real-life cases. For the first of these concerns, we extend the 'Temporal Logic of Security Actions' (S-TLA) language to describe a general system and introduce the property of 'action visibility'. We use this to create a model that represents one type of tampering, artefact destruction, and characterise it into four specific sub-types. For the second concern, we apply inductive reasoning to study the impact of artefact destruction on a generic system through comparing its state before and after the action. From this we characterise features of a system which has been subject to this tampering action. We present two brief case studies which illustrate how such features can be used to recognise whether tampering has occurred. Finally, we extend this technique to other types of artefact tampering to create a knowledge framework that can be used to aid the recognition of tampering activity.

1. Introduction

Digital forensic analysis is expected to develop a credible scientific theory that explains past events on systems based on the artefacts that can be retrieved (Gladyshev and Patel, 2004). Formal modelling has been used previously in digital forensics to provide such a credible, scientific basis for event reconstruction (e.g. Gladyshev and Patel, 2004; James et al., 2009; Marrington et al., 2010). These models aid mathematically well-founded logic, i.e. establishing *what* events occurred, the *order* they occurred and where possible *who* performed the action (Chabot et al., 2015).

However, several issues can frustrate this digital event construction process. Vanini et al. (2024) describe four such problems, including the passing of time, artefact contamination, insufficient knowledge of artefacts and the focus of this paper, artefact tampering. If artefact tampering is not recognised, there is a significant risk of data being misinterpreted in the rest of the investigation (Vanini et al., 2024). This is a critical issue for the field of digital forensics as legal systems depend on the presentation of authentic evidence (Schneider et al., 2022). A recent hearing by the Court of Arbitration for Sport (CAS 2020/O/6689 World Anti-Doping Agency vRussian Anti-Doping Agency, 2020) exemplifies how such activity can be used to attempt to undermine the legal process and alter the natural course of justice. In this judgement, CAS cited how 'advanced digital forensics' was integral to uncovering the extent of artefact tampering that had occurred.

To support efforts to recognise tampering during event reconstruction, there are two issues that we seek to address in this paper. First, the need to precisely understand what tampering is. Current understanding of tampering is based on informal natural language descriptions, such as provided by Harris (2006) and Kessler (2007). These are helpful to investigators as they outline different ways in which tampering can occur; however, they are not designed to be applied to models. They are imprecise in nature and can therefore be interpreted in a wide variety of ways. In this paper we address this imprecision by deriving a formal representation of tampering. We build on existing research by Rekhis and Boudriga (2012a, 2012b, 2010) who propose a model of a system

* Corresponding author. Milton Keynes, MK7 6AA, United Kingdom.
E-mail address: cjn.Correspondence@protonmail.com (C. Neale).

https://doi.org/10.1016/j.fsidi.2026.302147
Received 10 February 2026; Received in revised form 13 April 2026; Accepted 13 June 2026
Available online 23 June 2026

which incorporates the possibility of tampering based on temporal logic, using 'Temporal Logic of Security Actions (S-TLA). A glossary of terms we use and their notation for this formalism is contained in Appendix A.

The second issue we consider is how such models can be applied to reason about tampering and generate scientific theories regarding what happened. Recent research by Neale (2023) has shown that the most common methodology to recognise tampering is to apply deductive reasoning through the application of model-checking techniques. These are designed to provide conclusions that are logically provable. However, there are significant drawbacks to using this approach in practice, as noted by Gruber et al. (2023). First, analysing the state space of such models results in high time complexity. This severely limits applicability to more complex real-life cases. Additionally, inferring models that are faithful to the systems they are designed to represent is noted as a substantial problem, lacking known automation and relies on human reasoning to generate. We note that while Neale (2023) identified 27 published techniques, none of them had public implementations that were available to practitioners and none of them have been incorporated into common digital forensic toolkits such as Encase (OpenText Corporation) or X-Ways (X-Ways Software Technology).

Our approach in this paper is to consider inductive reasoning as an alternative approach to the practical problem of recognising tampering. This is the most common inference in digital forensics practice as it allows for reasoning in cases of uncertainty (Franqueira and Horsman, 2020). Unlike deductive reasoning, which is considered occasional in digital forensics practice, inductive reasoning has useful properties for forensic investigations, namely reaching conclusions which are probabilistic, defensible and ampliative (Pollitt et al., 2018).

We study how our representations of tampering actions can be used to make observations, or generalisations about the changes that occur on a system because of this type of adversarial activity. This enables us to codify the knowledge about a system that a practitioner requires to recognise the tampering action. We demonstrate how a simple feature analysis algorithm can be used for this purpose through two case studies. Our objective is to use the model to recognise occurrences of one type of tampering, artefact destruction, and generate credible theories about past events. We then extend this beyond artefact destruction to present a knowledge framework that aids the recognitions of all types of tampering.

The rest of this paper is structured as follows. In section 2, we present gaps in current research related to recognising tampering activity in digital forensics. In section 3, we present a representation of one type of artefact tampering (artefact destruction) through the application of the S-TLA formalism, aided by a running example. In section 4, these S-TLA representations are used to identify observable changes created by artefact destruction. Two short case studies are presented to outline this approach. In section 5, we extend the S-TLA representations to other types of tampering to create a knowledge framework for tamper recognition and detection. In section 6, we evaluate our approach against related work, before providing some conclusions and potential future work in section 7.

2. Recognising tampering: gaps in current research

To recognise a phenomenon (in this case tampering), Wieringa (2014, p.84) argues that it is strongly desirable to have a suitable conceptual framework that precisely describes what it is. However, tampering is currently represented inconsistently in the literature using subjective, natural language descriptions. This causes ambiguity which has the potential to undermine the ability of the forensic community to recognise tampered artefacts. Practitioners therefore rely on a curious and investigative mindset to implement a scientific method to track down inconsistencies that could be a result of tampering. Whilst this can be effective, it doesn't allow for a systematic approach to the issue in question.

For example, there is no clear agreement as to the high-level categories of tampering activity, with two main groups of four provided by Harris (2006) and Kessler (2007). However, these categorisations do not agree with each other, with Harris preferring to classify tampering by its impact on 'digital evidence' and Kessler choosing to base his on those provided in an early commercial presentation on 'anti-forensics'. In this paper, we prefer the Harris (2006) categorisation for the same reasons outlined in his original work, specifically that some of the alternative categories overlap, such as 'trail obfuscation' and 'attacking the forensic process and tools'. We largely remain consistent with this classification when discussing types of tampering. However, we have preferred the term 'artefact' over 'evidence' used in the original. This is because 'evidence' can imply that the object in question has probative value in substantiating a hypothesis of what has occurred, which will not always be the case.

Within individual types of tampering, there is however further inconsistency. We present 'artefact destruction' as an example. This has been described as '*wiping files*' by de Beer et al. (2015), as '*disposing of data securely*' by Zdzichowski et al. (2015) and as '*dismantling evidence or otherwise making it unusable to the investigative process by partially or completely obliterating it*' by Harris (2006). Each of these describes an aspect of artefact destruction (i.e. wiping, disposing and dismantling) and provides a broad understanding of how it might be achieved. However, there is variation even in this small sample of descriptions. For example, an action that partially obliterates some digital evidence is destruction for Harris (2006). Arguably, this would not be considered secure disposal by Zdzichowski et al. (2015) and may not even have any relation to with files, making it irrelevant to the de Beer et al. (2015) description. Additionally, it would be difficult to know whether the standard set by Harris (2006) of making this evidence unusable to an investigation had been achieved. It could be argued that even if the data was still 'useable', any attempt to disrupt this would also be tampering. These somewhat trivial examples demonstrate how such descriptions of tampering can create ambiguity which can only hinder the ability of the community to recognise tampering.

Modelling has been used previously as a tool for reasoning about past events. However, to apply this approach to recognise tampering we need a way to represent the tampering activities within the context of the formalism used. This is not possible with the type of natural language descriptions illustrated above. Several models for event reconstruction do exist, however few of them have explicitly considered tampering and none have sought to represent it explicitly. This is the case for many formalisms, including Finite State Machines (Gladyshev and Patel, 2004; Gruber et al., 2023), Witness Statements (James et al., 2009), Computer Profiling (Marrington et al., 2010), Linear-time temporal logic (Dewald, 2015) and action-trace update time thresholds (James and Gladyshev, 2014). In all cases, the methods are shown to be effective in reconstructing whether known types of events occur, which should make them applicable to the problem of recognising tampering. However, they all implicitly rely on the ability to precisely represent specific events in a formalism to test whether they occurred in a particular instance. This means that they are currently unsuitable to the task of recognising tampering.

The one notable exception is Rekhis and Boudriga (2012a, 2012b, 2010) who propose a model of a general system based on the Temporal Logic of Actions (originally Lamport, 1999). This model of a system is referred to as S-TLA and is designed specifically for reasoning about tampering (referred to in these works as 'anti-forensics'). It is therefore particularly applicable to our interests in this paper. The work in Rekhis and Boudriga (2012a) also provides some initial insight into how tampering actions may manifest in this model. This is the most precise understanding of tampering that we have discovered in our literature search. However, the authors don't explicitly represent the various tampering actions, nor do they enumerate any variations on the general types of tampering action as this is not the focus on their work. In this paper we extend this proposed model to create our representations of types of artefact destruction in section 4, artefact source elimination in

section 5, and other tampering types in Appendix B. However, the case studies presented in these papers for investigating anti-forensics assume that the totality of potential anti-forensic attacks are already known. This restricts the ability for the techniques as described to be practically applied and is why we have chosen to take a different approach in our reasoning.

Using modelling to support event reconstruction can generate a credible scientific theory about what has occurred; however, there are limitations to this approach in practice. A typical application is to use deductive reasoning to generate conclusions which are mathematically provable. This is most commonly achieved by using model-checking algorithms to evaluate possible previous system states (Neale, 2023). James and Gladyshev (2014) argue that applying model-checking via a formalism such as temporal logic allows for event reconstruction to be reduced to a state-space exploration problem. However, the state-space of even the simplest real-world systems are considered computationally impractical (James and Gladyshev, 2014). This is a well-documented issue in the literature known as state-space explosion problem (Soltani and Seno, 2019). This is seen practically in the work of Gruber et al. (2023) who found that the high-time complexity of calculating evidence sets was problematic, alongside the technical limitations in the sizes of models that could be processed with expanding space-state. The authors additionally noted that creating faithful models of systems under investigation is challenging and requires a manual and time-consuming approach.

In this paper we first address the gap relating to a lack of precise representation of tampering. We achieve this by constructing representations of artefact tampering by extending the S-TLA model. We then apply a form of inductive reasoning to understand the impact that artefact tampering has on a system. By comparing system state before and after the tampering activity we characterise features of the artefacts that will exist on a system which has been subject to this action. This allows us to identify properties of tampering which can then be applied to different real-world situations. We apply feature analysis to classify an input artefact as tampered or non-tampered. We outline two brief examples in section 4 which illustrate how such features can be generated and applied to the task of recognising whether artefact destruction has occurred. When this approach is extended to other types of tampering, we create a framework that describes the knowledge needed to recognise all types of tampering during event reconstruction. This framework is presented in section 5.

3. A formal representation of artefact destruction

To address the lack of precise representations of tampering we constructed representations in symbolic logic which extend from an existing formalism we have selected, namely S-TLA. Before we outline this work, we provide an overview of a running example used to map concepts from the theoretical model to real-world forensic practice. This example provides some context for how the theoretical aspects of our work relate to real forensic investigations.

3.1. Running example – forensic analysis of door entry logs

The 'Very Simple Door Lock' is an example system for controlling a door. To lock or unlock the door, a smart card is presented to an electronic reader for authentication. A database is used to keep track of a secret token (unique to each smart card) and the user to which it is assigned. The database also contains their role and the last time each available system 'mode' was modified. Once a valid user is authenticated, they can toggle the status of the door lock. Either action causes an internal variable 'LOCK_STATUS' which can have one of the two states 'LOCKED' or 'UNLOCKED'. The system also keeps track of physical attributes, such as the angle by which the door is opened. Two files on this system have relevance to a forensic investigation.

- A logging process which writes output to the file 'log.txt'. An example of such a file is provided in Fig. 1 which contains log entries for a particular execution of the door lock
- A database process which maintains a table of users and a table of status modes in the database 'data.db'. An example of such a database is provided in Fig. 2 which shows the system in use with 3 users

This example has some properties which make it useful for our purpose in studying tampering in a forensic context. First, we are not provided with all data that is potentially accessible within the system. For example, the angle by which the door is opened is tracked by the system but not recorded in either of the two files that are available. Second, real actions undertaken by humans on the system can be reflected in data across the two files of the system that are forensically interesting. This includes activities such as the registration of a new user, which can be seen in data from both sources. In later sections we will discuss how these properties enable a more precise reasoning about tampering through our model.

As we conclude this section, we propose a question for the reader to consider. Can artefact destruction be recognised in the data provided in Figs. 1 and 2? As the data presented is studied, readers may formulate hypotheses about what may be considered 'out of place' or unusual for the system based on the details we have provided. As this is a paper on tampering, it may be reasonable to assume that we have included an example here. However, without a suitable framework to describe what tampering is, it is challenging to be precise about whether it has occurred. Furthermore, the task of authenticating the data as being free from tampering is equally challenging. Additionally, we invite readers to reflect on the type of logic they have used to formulate hypotheses about tampering in this case and whether they have found deductive or inductive more natural.

We will return to this question in the brief case study in section 4.2.

3.2. S-TLA as a means for modelling a general system

To represent tampering, we first needed to select an appropriate formalism to use. One option we encountered during a search of the literature used a formalism for describing a general system, based on the Temporal Logic of Security Actions, S-TLA. This has been developed by (Rekhis and Boudriga, 2010, 2012a, 2012b; 2012a) and extended from the original Temporal Logic of Actions (TLA) created by Lamport (1999). The authors had developed this model with the objective of using it for the purposes of identifying tampering activity. S-TLA provided a formalism which can be utilised to describe general systems and describe a system state before and after an action occurs using some form of temporal logic. They can therefore be used to reason about the impact of a tampering action on such a system. In this section, we briefly recap the aspects of this model which we build upon in this paper. For a more detailed explanation of S-TLA we refer the readers to the original research. For a glossary of relevant terms for easy reference, see Appendix A. Several alternative choices of notation exist that could potentially have been used, including Finite State Machines (e.g. Carrier and Spafford, 2004), Transition Systems (e.g. Soltani and Seno, 2019) and Simplified Event Calculus (e.g. Willassen, 2008). In this paper we provide a representation using a single notation in S-TLA and leave other representations as potential future work.

In S-TLA, a **general machine** is defined as a **set of variables** $\mathscr{V} = \{v_0, \ldots, v_n\}$. The **state** of this machine is given by the **set of values of these variables** $s = \{v_0[s], \ldots, v_n[s]\}$. An **action** represents the **transition from one state to the next**. Formally, an action $A \in$Acts and $A(s,t)$ is true if and only if A is enabled on state s and its execution produces the new state t. The set 'Acts' is the set of all possible actions that can occur. The **execution** of a system is defined as the **sequence of states** that occur. This is denoted as $\varpi = (s_0, \ldots, s_n)$.

In the running example, $\mathscr{V}$ consists of variables such as the door lock status, the orientation of the door and the current authenticated user,

```
1  2023-10-11 08:32:19 UTC       Door is UNLOCKED by User 1 (Alice)
2  2023-10-11 08:32:20 UTC       Door is open
3  2023-10-11 08:32:34 UTC       Door is closed
4  2023-10-11 08:32:36 UTC       Door is LOCKED by User 1 (Alice)
5  2023-10-11 09:18:43 UTC       Door is UNLOCKED by User 2 (Bob)
6  2023-10-11 09:18:43 UTC       Door is open
7  2023-10-11 09:19:15 UTC       Door is closed
8  2023-10-11 09:41:03 UTC       Door is open
9  2023-10-11 09:41:15 UTC       Door is closed
10 2023-10-11 09:41:22 UTC       Door is LOCKED by User 1 (Alice)
11 2023-10-11 10:02:16 UTC       Door is open
12 2023-10-11 10:02:32 UTC       Door is closed
13 2023-10-11 11:00:42 UTC       System set to Administrative mode
14 2023-10-11 11:02:12 UTC       New user registered as User 3 (Charlotte)
15 2023-10-11 11:02:31 UTC       System set to User mode
16 2023-10-11 11:02:48 UTC       Door is UNLOCKED by User 1 (Alice)
17 2023-10-11 11:02:51 UTC       Door is LOCKED by User 1 (Alice)
18 2023-10-11 11:03:46 UTC       Door is UNLOCKED by User 3 (Charlotte)
19 2023-10-11 11:03:59 UTC       Door is open
20 2023-10-11 11:04:12 UTC       Door is closed
21 2023-10-11 11:04:16 UTC       Door is LOCKED by User 3 (Charlotte)
22 2023-10-11 13:56:03 UTC       Door is set to OVERRIDE
```

Fig. 1. Example of log.txt file.

```
1  SELECT * FROM users;
2
3  | id  | token   | username     | role          |
4  | 1   | a8ef541 | Alice        | Administrator |
5  | 2   | 61a8bc3 | Bob          | Employee      |
6  | 3   | 73fe9ab | Charlotte    | Employee      |
7
8  SELECT * FROM mode;
9  | system_mode     | last_updated        |
10 | User            | 2023-10-11 11:02:31 |
11 | Administrative  | 2023-10-11 11:00:42 |
12 | OVERRIDE        | 2023-10-11 13:56:03 |
```

Fig. 2. Example of data.db file.

among many others. As a forensic investigator, we are unlikely to be able to comprehend every possible system variable and therefore we do not attempt to define the entire set for this example. Additionally, we do not attempt to define the entire set 'Acts' as this is also unlikely to be known by a forensic investigator.

A system also has a set of '**Observers**'. Each observer can access a subset of $\mathscr{V}$. These are defined in the set $O = \{o_0, \ldots, o_k\}$. The observation function $\text{obs}(s_i) = v_j[s_i]$if the variable v_i can be observed at state s_i by an observer in O, otherwise it is ignored. A **trace** is defined as the sequence $tr(\varpi) = (\text{obs}(s_0), \ldots, \text{obs}(s_m))$. This is the **sequence obtained from** $\varpi$where each element s_i is replaced with $\text{obs}(s_i)$. **Individual observers** within O act on this trace function to **create an artefact** $a(tr(\varpi))$ if a particular predicate it true, i.e. if $\varphi(tr(\varpi)) = \text{true}$ for some predicate φ and observer o_a. It is down to each individual observer to define zero or more predicates that result in the artefacts that are created by them. In our running example, O contains two elements, the database process and the logging process. A predicate for the logging process might state that if the door is unlocked, then an entry is written to log.txt, such as line 1 in Fig. 1.

An **'artefact container'** is a **set of artefacts produced by an observer** as a result of an execution, given as $ac(\varpi) = \{a(tr(\varpi)^i) \mid i \in [0.n]\}$. $A_{\bar{s}}$ is the set consisting of all the elements within artefact containers, known as the **artefact set**. In the original research, a distinction is made between artefacts that could have been tampered with (using the $A_{\bar{s}}$ notation) and the artefacts that can be 'trusted' (which simply uses A_s). We argue that a better approach is to consider that all artefacts may not be trustworthy, as we previously advocated in Neale et al. (2022). Therefore, we consider all artefacts to be part of the former and we have retained this notation for consistency with the original work. This artefact set is the set of objects we consider as being available to a digital forensic investigator. Note that we have substituted the term 'evidence' in the original for the term 'artefact' throughout this paper. This is because any digital object or trace does not become evidence until a decision is taken that it is relevant to the case under investigations (Margot, 2011). This is not the focus of our study, and we have chosen to retain the term artefact for clarity.

In our running example, the file log.txt is an example of an artefact container, as is the data.db database. These contain the individual artefacts, i.e. the lines in the log files and the records in the database. The artefact set is the set consisting of all artefacts in all artefact containers (i.e. all entries in log.txt and all records in data.db).

3.3. Extending S-TLA to introduce the concept of visibility for actions

For our purposes, we extend a particular concept from the original research to incorporate an additional property, the visibility of an action on $A_{\bar{s}}$. The original work considered the visibility of individual variables, but we now consider this same property for actions. When an action occurs, the new state is appended to the sequence ϖ. It follows that newly observed variables are appended to the trace, i.e. $\text{obs}(s_n)$ is appended to $tr(\varpi)$ to become $tr'(\varpi)$. Each observer in O can then act on this to potentially add new artefacts to their own artefact containers and in doing so, extend the overall artefact set, which becomes $A_{\bar{s}}'$.

Definition: The visibility of an action X is given as δ_X where:

$$\delta_X = A_{\bar{s}}' - A_{\bar{s}}$$

Therefore, an action is invisible if $\delta_X = \{\ \}$

The **visibility** of an action is **the subset of elements of the artefact set that are generated by that action.** In our running example, the action of unlocking the door by presenting a smart card adds one artefact, a single entry to the log.txt artefact container. Therefore, δ_X contains this single element. In contrast, setting the system to 'Administrative' mode writes to log.txt (line 13 in Fig. 1) and updates a row in data.db (line 11 of Fig. 2). Both actions are visible, whereas the

action of swinging the door once unlocked and opened (i.e. changing the angle the door is opened) is invisible. This is because this action does not create any additional elements in the artefact set based on the observers defined in the example.

3.4. Using S-TLA to represent artefact destruction

We are seeking to precisely state what tampering is to examine how it can be recognised during investigations. Using the work from previous sections to summarise and extend the S-TLA formalism, we now propose an initial representation of one type of tampering, artefact destruction. We will use these definitions in section 4 to show how they can be applied to the task of designing an algorithm to identify tampering during investigations.

To do so, we work from what we consider to be a useful description of artefact destruction from widely cited existing literature based on the work of Harris (2006). This describes artefact destruction as:

Dismantling evidence or otherwise making it unusable to the investigative process by partially or completely obliterating it.

In terms of our model, we can interpret this as modifying $A_{\bar{s}}$ by removing all specific artefacts relating to a previous action that occurred, X. Rekhis and Boudriga (2010) describe this as an anti-forensic attack to "hide the effect of the action X" in the artefact set.

We therefore initially propose the following representation of artefact destruction:

Definition: An action $T(s_i,s_{i+1})$ is artefact destruction if as a result:

$$A_{\bar{s}} \rightarrow A'_{\bar{s}} \text{such that } A'_{\bar{s}} \subset A_{\bar{s}} \text{ and } \forall\, d \in \delta_X,\ d \notin A'_{\bar{s}} \text{for some action } X$$

Note two constraints specified in the initial representation. The first is that the new artefact set is a strict subset of the original. It follows that T must therefore be invisible. This concept is not explored in the original description. However, in our professional experience, we can certainly conceive of scenarios where this is not possible to achieve for an adversary. For example, an adversary may wish to destroy an incriminating registry key on a Windows operating system, many of which require SYSTEM level privileges. If the adversary has compromised a user account, this would not be possible unless they were also able to escalate their privileges to this level.

The second constraint specifies that all elements of δ_X are removed from the artefact set. We introduce the terminology completeness to describe this. In the original description this matches with the phrase 'complete obliteration' but does not account for the case of 'partial obliteration'. Therefore, we require a way to reason about these scenarios. As a result, we expand on the initial representation to provide four specific sub-types of artefact destruction.

The first accounts for the case where both constraints are met. The tampering action adds no elements to the artefact set and where all the elements associated with a previous action X from the artefact set are removed. We call this invisible incomplete artefact destruction (ICAD).

Definition (ICAD): An action $T(s_i,s_{i+1})$ is invisible complete artefact destruction if as a result:

$$A_{\bar{s}} \rightarrow A'_{\bar{s}} \text{such that } A'_{\bar{s}} \subset A_{\bar{s}} \text{ and } \forall\, d \in \delta_X,\ d \notin A'_{\bar{s}} \text{for some action } X$$

Example: Consider an action X that involves a door unlock, open, close and lock. In this case, δ_X consists of four elements, each of which is an entry in log.txt (such as lines 1-4 in Fig. 1). A tampering action T that removes all four entries of log.txt would satisfy the completeness constraint in that there are no elements of δ_X in the new artefact set $A'_{\bar{s}}$. If it also did not add any lines to log.txt or change any of the rows in data.db it would also be invisible as the new artefact set would be a subset of the former, $A'_{\bar{s}} \subset A_{\bar{s}}$. This would satisfy the conditions for the tampering action to be ICAD.

As a side note, we hypothesise that in practice, meeting both conditions to satisfy ICAD could be very difficult to achieve in practice. On the completeness aspect, modern desktop operating systems such as Windows and macOS produce large numbers of artefacts in myriad locations. This is a direct challenge to the 'completeness' aspect. Furthermore, these systems typically have large numbers of 'observers' for the purposes of telemetry and instrumentation, resulting in these artefacts being potentially available. Removing all of these while leaving no visible additional artefacts requires complete knowledge of OS internals and high levels of privilege. However, not all systems are necessarily as complicated. Others, such as those powering IoT devices may provide an opportunity for an adversary to conduct tampering that is invisible and complete.

We now account for the case where the first constraint cannot be met, i.e. tampering action itself creates additional entries in the artefact set. If all elements of a previous action X are still removed, then this is visible complete artefact destruction (VCAD).

Definition (VCAD): An action $T(s_i,s_{i+1})$ is visible complete artefact destruction if as a result:

$$A_{\bar{s}} \rightarrow A'_{\bar{s}} \text{such that } \delta_T \neq \{\} \text{ and } \forall\, d \in \delta_X,\ d \notin A'_{\bar{s}} \text{for some action } X$$

Example: Consider an action X that involves a door unlock, open, close and lock. In this case, δ_X consists of four elements, each of which is an entry in log.txt (such as lines 1-4 in Fig. 1). A tampering action T that removes all four entries of log.txt would satisfy the completeness constraint in that there are no elements of δ_X in the new artefact set $A'_{\bar{s}}$. However, if T requires administrative access to the system, then this creates new entries in the artefact set (see line 13 in Fig. 1 for the impact on log.txt and line 11 in Fig. 2 for the impact on data.db). If T does not also remove both entries from the new artefact set $A'_{\bar{s}}$, then $\delta_T \neq \{\}$. Therefore, T would be an example of VCAD.

We next account for the case where the second constraint cannot be met, i.e. tampering action is invisible but does not remove all elements of a previous action X. This is invisible incomplete artefact destruction (IIAD).

Definition (IIAD): An action $T(s_i,s_{i+1})$ is invisible incomplete artefact destruction if as a result:

$$A_{\bar{s}} \rightarrow A'_{\bar{s}} \text{such that } A'_{\bar{s}} \subset A_{\bar{s}} \text{ and } \exists\, d \in \delta_X,\ d \in A'_{\bar{s}} \text{for some action } X$$

Example: An action X to put the system in 'Administrative' mode will update the data.db mode table with a timestamp (such as Line 11 of Fig. 2) and create an entry in log.txt which indicates 'Administrative' mode has been started (such as Line 13 of Fig. 1). In this case δ_X contains these two elements. A tampering action T that can remove one of these entries results in a new artefact that contains some elements of the original δ_X, i.e. is incomplete. If T does not create any additional entries in log.txt and data.db, then the new artefact set is still a subset of the original and the constraints for IIAD are met.

Finally, we account for the case, where both constraints cannot be met. The tampering action is therefore visible and incomplete, making this visible incomplete artefact destruction (VIAD).

Definition (VIAD): An action $T(s_i,s_{i+1})$ is visible incomplete artefact destruction if as a result:

$$A_{\bar{s}} \rightarrow A'_{\bar{s}} \text{such that } \delta_T \neq \{\} \text{ and } \exists\, d \in \delta_X,\ d \in A'_{\bar{s}} \text{for some action } X$$

Example: Consider an action X that involves a door unlock, open, close and lock. In this case, δ_X consists of four elements, each of which is an entry in log.txt (such as lines 1-4 in Fig. 1). A tampering action T that only removes some of these four entries results in elements of δ_X being present in the new artefact set $A'_{\bar{s}}$. If T also required 'Administrative' mode, that would create additional entries in $A'_{\bar{s}}$ such as updating a timestamp in data.db (such as line 11 in Fig. 2). If these remain in the artefact set after T, then T is both visible and incomplete, denoted as VIAD.

4. Using S-TLA representations to recognise artefact destruction

In the previous section we proposed S-TLA representations of artefact destruction that characterise the nature of this type of action on a

Table 1
Observable changes to artefact set caused by artefact destruction, along with knowledge needed to identify.

Type of tampering action (T)	Difference between ϖ and ϖ'	Knowledge required to differentiate	Real-world explanation
ICAD	None	N/A	N/A
VCAD	Elements of δ_T in the artefact set	Elements of δ_T	The tampering action itself leaves a trace
IIAD	Elements of δ_X in the artefact set	Elements of δ_X which appear in the artefact set without the whole of δ_X	Some action can be inferred but only some of the expected traces can be found
VIAD	Elements of δ_T in the artefact set Elements of δ_X in the artefact set	Elements of δ_T Elements of δ_X which appear in the artefact set without the whole of δ_X	The tampering action itself leaves a trace OR Some action can be inferred but only some of the expected traces can be found

system. In this section we will demonstrate how these representations can be used to explain how a system is changed by the four types of artefact destruction. We do this by comparing an S-TLA system execution before and after each action and analysing the difference in the resulting artefact sets. Specific details are provided in section 4.1 and this is followed by two examples in section 4.2 and4.3 which demonstrate how this could be applied in practice.

4.1. Using the model to specify observable changes made by artefact destruction

Consider a system execution ϖ. We compare this to the execution of ϖ' which extends ϖ by a single tampering action (a type of artefact destruction). We compare ϖ and ϖ' in Table 1 to find the difference in the generated artefact sets. This is the observable change that occurs on the system because of the artefact destruction action. We also record the knowledge about the system an investigator or forensic tool would need to distinguish between the two executions by inspecting the artefact set.

Artefact destruction can therefore be recognised in two ways. First, we can identify artefacts associated with a known tampering action T (elements of δ_T). We call this 'Identifying Attacks' with a label of 'AT'. This is comparable to signature-based approaches to detecting known malicious activity. Alternatively, we can identify artefacts which infer some action A but where only some of the expected artefacts of A can be found (elements of δ_X which appear in the artefact set without the whole of δ_X). This is a heuristic-based approach to detecting divergence from normal behaviour. In this case, we need to understand the totality of artefacts created when A occurs on a particular system to recognise deviation. Let us call this 'Identifying Abnormality', labelled as 'AB'. Note that if an artefact destruction action is both invisible and complete (ICAD), then there is no means of identification.

4.2. Example 1 – recognising artefact destruction in the running example

Now that we can formally represent tampering actions, we could use existing model-checking techniques to study the state space of a machine specified in S-TLA and determine whether a tampering action occurred. However, the work of other authors has shown that these techniques have significant limitations. For the running example, we would need to create a working model of the system in the relevant formalism. Additionally, we would also need to be able to define every system variable and every potential action in 'Acts'. Therefore, we propose an alternative approach which is designed to produce an approximation of whether tampering has occurred based on the features of system execution that relate to artefact destruction from the previous section. Now that we can specify the knowledge required to differentiate between ϖ and ϖ', we can use this to generate the features can potentially indicate tampering on a given system. An initial version of this approach is given in Algorithm 1:

```
// Step 1: Generate features for the system which potentially indicate tampering
Feature_List = [feature_1, … , feature_n]

// Step 2: Configure Parameters
TAMPERED ← FALSE
FEATURES ← 0
THRESHOLD ← 0.5 // or selected appropriate value

// Step 3: Inspect each feature
for each FEATURE in Feature_List do
   if FEATURE does not indicate tampering then
      FEATURES ← FEATURES + 1
   end if
end for

// Step 4: Calculate FEATURES average
NORMALITY_SCORE ← FEATURES / length(Feature_List)

// Step 5: Compare against threshold
if NORMALITY_SCORE < THRESHOLD then
   TAMPERED ← TRUE
end if

// Step 6: Report result
return TAMPERED
End Algorithm
```

In this paper, we specify this algorithm conceptually and leave its

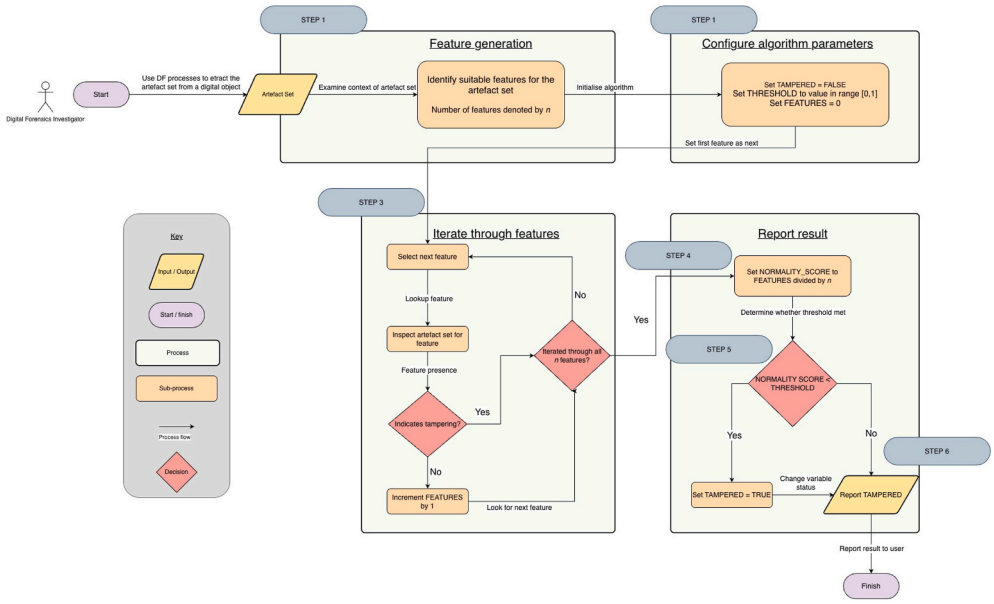

Fig. 3. Process diagram for Algorithm 1.
Note that features here are specified as a binary condition, where 'passing' is indication of non-tampering, whereas 'failing' is an indication of tampering, specifically artefact destruction.

implementation as an automation to future work. For example, we generate features based on observations about how the systems and files in the examples operate. Operationalising this aspect of Algorithm 1 for greater scalability and applicability is not considered here. Furthermore, the values set in Algorithm 1 for THRESHOLD (i.e. the threshold for classification into tampered or non-tampered as reflected by the TAMPERED variable) aim to illustrate the approach. Determining a suitable value for optimising performance of this approach in different contexts will be conducted in future research activities.

A process diagram for Algorithm 1 is shown in Fig. 3.

We now return to the question initially posed in section 3.1 and apply this algorithm to our running example. Some example features for step 1 are given in Table 2.

Figs. 5 and 6 show the same data from the running example (Figs. 2 and 3 with additional annotations which relate to these features.

Step 2 relates to setting variable and constant values. For our example, we set THRESHOLD to be 0.5. Then in step 3, we inspect the artefact set to find occurrences of the features in question. For feature 1, we can see that the system is set 'Administrative' mode (yellow box in Fig. 5), so FEATURES remains at 0. Looking at feature 2, we also note that we are unable to see the actions that follow this time as there are no logs generated after 2023-10-12 03:12:09. We again do not increment FEATURES. Finally, for feature 3, we see a few instances of the door being unlocked, opened, shut and locked again. The instance in the red box of Fig. 4 conform to expectations, however the blue box (lines 5-8 of Fig. 4) does not. FEATURES therefore remains at 0. This concludes step 3.

In step 4, we divide FEATURES by the number of features, in this case $0/3 = 0$. As this is less than THRESHOLD (0.5), we set TAMPERED = TRUE (step 5) and report this as output (step 6). We can additionally be more precise in our characterisation of the potential tampering. We know that the potential tampering action is visible (line 11 of Fig. 5) that at least some artefacts associated with a previous action were not fully destroyed (lines 5-8 of Fig. 4). Therefore, this is an example of visible incomplete artefact destruction (VIAD).

This example illustrates how the S-TLA representations of artefact destruction can be applied to the practical problem of identifying tampering. Algorithm 1 relies on the specification of suitable features to correctly classify between tampered and non-tampered artefact sets. These features are generated based off the two conditions used to represent artefact destruction in the S-TLA model, i.e. visibility and completeness. Many more features could be specified, even for a relatively straightforward system like the one derived in our running example, however we have only chosen a few to illustrate the potential practical approach that could be used. Maturing this approach to scale the number of features and tampering types is left for future work. While the S-TLA model of tampering presented does not depend on the specific technology implemented, the specification of suitable features for identifying abnormalities (AB) and attack (AT) does. Our ongoing research is currently focussed on automating this approach and understanding how technologies such as Machine Learning can enhance the practicality of what is outlined in this paper. This a subject of ongoing research within the digital forensics community as shown in recent research by Michelet et al. (2026).

Before we extend our model to other types of tampering beyond artefact destruction in section 5, we now briefly present a second example. For this, we have chosen to use SQLite files as our artefact in question to show that the conditions from the S-TLA model can be applied to an artefact commonly found in real-world investigations.

4.3. Example 2 – recognising artefact destruction in SQLite files

We present one further brief case study. SQLite files are common on a myriad of platforms and provide the ability to enable lightweight database storage (Nemetz et al., 2018). Due to their prevalence, they are a common source of artefacts in digital forensic investigations. In this example we investigate a SQLite database file which is used to record transactions in a financial application. The purpose of this example is to show that the S-TLA representations of artefact destruction can be used to derive features of real artefacts that can be used by Algorithm 1 to classify between tampered and non-tampered artefacts.

The file, named 'transactions.db' contains two tables, 'transactions' and 'audit'. We see a subset of the content of 'audit' in Fig. 6 and of 'transactions' in Fig. 7. An additional record is shown in Fig. 8.

Applying algorithm 1 again, we identify the features outlined in Table 3 and focus particularly on step 3. No tampering is indicated from feature 1, but we can see that some method was used to make records immutable in the transactions table (feature 2). This is indicated in lines

Table 2
Features of potential tampering in running example.

Feature	Descriptions	Label
1	Efforts to manipulate artefacts on the system require it to be in 'Administrative' mode	AT
2	When the system is in 'Administrative' mode, the actions that follow should indicate legitimate system management, otherwise it might be tampering	AB
3	When a user goes through the door, the events should reflect the following order: UNLOCK, door opens, door closes, LOCK	AB

5 and 6 of Fig. 6. Additionally, there are missing transaction ID's in Fig. 7 (feature 3). Furthermore, Fig. 8 shows that for at least one transaction, the details show as blank (feature 4). Therefore, in our algorithm, the final value of FEATURES would be 3, with step 4 resulting in a value of $1/4 = 0.25$, which is again less than THRESHOLD. As a result, the algorithm reports a final output of TAMPERED.

Additionally, we can be more specific with regards to what tampering might have occurred. As 'secure delete' is set we can assume the records are not recoverable. However, the ID values of surrounding records have not been adjusted to compensate, therefore the missing transactions in Fig. 7 are a result of visible complete artefact destruction (VCAD). In this case the 'visible' property is caused by the discrepancy in database row IDs which are designed in the case of this example to be sequential. This illustrates how even relatively simple design features can make conducting 'invisible' types of tampering difficult to achieve in practice.

For the destruction of the value of the 'Details' in ID 36821 (Fig. 8), we can see other artefacts related to the original action (i.e. the date, time etc.) meaning that the destruction was incomplete. However, based on these features, the actual tampering action that caused this is not visible in any way. Therefore, this is invisible incomplete artefact destruction (IIAD). Our expectation is that this finding would trigger further activities to investigate this hypothesis further. In this case, inspection of the record manually using a hex editor shows some of the contents of this partially destroyed record, which is seen in Fig. 9. This is evidence of use of a technique described by Schmitt (2018) in which a NULL byte (indicated by the red box) prevents the string from being displayed by SQLite.

This example supplements the previous one by demonstrating how a tampering action proposed in peer-reviewed research (Schmitt, 2018) for SQLite files can be recognised by Algorithm 1 through features derived from our S-TLA model. Again, we have chosen only a few features for the purpose of illustration in this paper. However many more features of SQLite files could be used to attempt to identify artefact destruction (and potentially other tampering types). This includes features such as the internal workings of the various SQLite journal options (e.g. rollback journals or the 'Write Ahead Log' – WAL) or other mechanisms such as the integrity PRAGMA capability to detect inconsistencies.

From these brief examples, defining suitable features for observing tampering across a range of different systems and artefacts is key to success of the approach we have outlined. Based on the model from section 4, we can see that this will require both a forensic understanding of 'normal behaviour' of these systems and artefacts (AB) as well as known attacks that can occur (AT). This is not considered a limitation as this is a requirement for any practitioner analysing a system to identify tampering. The power of this approach is that this knowledge can be encoded by experts in one type of artefact. This can then be used as a tool to flag tampering to those with less knowledge or experience of a given artefact.

Such known attacks may also need to be thoroughly tested and observed in various scenarios. Providing an empirical evaluation of this approach to any specific artefact type is beyond the scope of this paper and the subject of ongoing research.

```
 1  2023-10-11 08:32:19 UTC       Door is UNLOCKED by User 1 (Alice)
 2  2023-10-11 08:32:20 UTC       Door is open
 3  2023-10-11 08:32:34 UTC       Door is closed
 4  2023-10-11 08:32:36 UTC       Door is LOCKED by User 1 (Alice)
 5  2023-10-11 09:18:43 UTC       Door is UNLOCKED by User 2 (Bob)
 6  2023-10-11 09:18:43 UTC       Door is open
 7  2023-10-11 09:19:15 UTC       Door is closed
 8  2023-10-11 09:41:03 UTC       Door is open
 9  2023-10-11 09:41:15 UTC       Door is closed
10  2023-10-11 09:41:22 UTC       Door is LOCKED by User 1 (Alice)
11  2023-10-11 10:02:16 UTC       Door is open
12  2023-10-11 10:02:32 UTC       Door is closed
13  2023-10-11 11:00:42 UTC       System set to Administrative mode
14  2023-10-11 11:02:12 UTC       New user registered as User 3 (Charlotte)
15  2023-10-11 11:02:31 UTC       System set to User mode
16  2023-10-11 11:02:48 UTC       Door is UNLOCKED by User 1 (Alice)
17  2023-10-11 11:02:51 UTC       Door is LOCKED by User 1 (Alice)
18  2023-10-11 11:03:46 UTC       Door is UNLOCKED by User 3 (Charlotte)
19  2023-10-11 11:03:59 UTC       Door is open
20  2023-10-11 11:04:12 UTC       Door is closed
21  2023-10-11 11:04:16 UTC       Door is LOCKED by User 3 (Charlotte)
22  2023-10-11 13:56:03 UTC       Door is set to OVERRIDE
```

Fig. 4. Annotated example of log.txt file.

```
SELECT * FROM users;

| id  | token   | username    | role          |
| 1   | a8ef541 | Alice       | Administrator |
| 2   | 61a8bc3 | Bob         | Employee      |
| 3   | 73fe9ab | Charlotte   | Employee      |

SELECT * FROM mode;
| system_mode     | last_updated        |
| User            | 2023-10-11 11:02:31 |
| Administrative  | 2023-10-12 03:12:09 |
| OVERRIDE        | 2023-10-11 13:56:03 |
```

Fig. 5. Annotated example of data.db file.

```
sqlite> SELECT * from audit;
| ID |        Date         |                    Message                    |
| 1  | 2024-06-19 18:23:09 | 'Secure Delete' option enabled on database    |
| 2  | 2024-07-01 00:00:03 | Database backup completed successfully        |
| 3  | 2024-08-01 00:00:09 | Database backup completed successfully        |
| 4  | 2024-09-01 00:00:11 | Database backup completed successfully        |
| 5  | 2024-09-04 08:17:03 | Database lock removed - transactions are no longer immutable |
| 6  | 2024-09-04 09:02:14 | Database locked - transactions are immutable  |
| 7  | 2024-10-01 00:00:11 | Database backup completed successfully        |
```

Fig. 6. Contents of audit table.

```
sqlite> SELECT * FROM transactions WHERE Date like '%2024-09-0%';
|  ID   |    Date    |   Time   |              Details              |
| 33842 | 2024-09-02 | 10:01:42 | 31,09 received from ACME Ltd      |
| 33843 | 2024-09-02 | 10:39:02 | 22,09 paid to General Consultancy |
| 33845 | 2024-09-02 | 12:38:17 | 210,04 paid to Business Catering Group |
| 33847 | 2024-09-02 | 15:09:12 | 18,04 paid to CleanersRUs         |
```

Fig. 7. Contents of transactions table.

5. Extending the model to other types of tampering

In this section we expand our application of S-TLA from section 3.4 and apply it to additional types of tampering. In doing so, we provide a representation of these other types of tampering actions in S-TLA. We apply the same logic outlined in Table 1 to provide a comprehensive understanding of the observable impact of these tampering actions. In the previous section we used two observations (labelled AT and AB) to define features of an artefact set which were then used to identify tampering using Algorithm 1. Extending our S-TLA model to other types

```
sqlite> SELECT * FROM transactions WHERE ID = 36821;
|  ID   |    Date    |   Time   | Details |
| 36821 | 2024-09-13 | 11:04:13 |         |
```

Fig. 8. Additional record in transactions table.

Table 3
Features of potential tampering in transactions.db

Feature	Descriptions	Label
1	Setting the table to use 'secure delete' is normal for this application	AT
2	Transactions should be immutable	AT
3	All transactions should be recorded	AB
4	Transactions must include details which include the amount of the transaction, whether it was 'paid' or 'received' and the name of the other party	AB

of tampering produces a total of 6 such labels that can be used for this purpose. The details of this are provided in section 5.2.

5.1. Representing additional types of tampering

First, we demonstrate that the same logic applied in section 3.4 for artefact destruction can be used to define S-TLA representations of other types of tampering. We take the example of artefact source elimination. This is described in Harris (2006) as:

Neutralizing evidentiary sources [so that there is] *no need to destroy evidence since it is never created.*

In terms of S-TLA, this involves removing observer elements in the set O to prevent relevant artefacts from being added to the artefact set for some future actions. Note here that for artefact destruction, we can infer that the tampering action T occurs after the action X that T is trying to destroy the record of. For artefact source elimination, the opposite is true, and T occurs before X. Furthermore, the two conditions of visibility and completeness also apply for this type of tampering.

Therefore, we can represent an artefact source elimination action in S-TLA that is invisible (creates no additional artefact entries) and complete (in preventing all elements of a future action X from being added to the artefact set).

Definition (ICASE): An action $T(s_i,s_{i+1})$ is invisible complete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T = \{\}$ and $\delta_X = \{\}$ for some future action X

If the tampering action itself leaves a trace, then the $\delta_T = \{\}$ condition is no longer true, and the action is visible (but still complete).

Definition (VCASE): An action $T(s_i,s_{i+1})$ is visible complete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T \neq \{\}$ and $\delta_X = \{\}$ for some future action X

Conversely if the tampering action remains invisible, i.e. $\delta_T = \{\}$ but not all the elements of a future action X are prevented from being added to the artefact set, then we have the action as invisible and incomplete.

Definition (IIASE): An action $T(s_i,s_{i+1})$ is invisible incomplete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T = \{\}$ and $\delta_X \neq \{\}$ for some future action X

Finally, if both conditions from the first definition are false, then we have the case of the tampering action being both visible and incomplete.

Definition (VIASE): An action $T(s_i,s_{i+1})$ is visible incomplete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T \neq \{\}$ and $\delta_X \neq \{\}$ for some future action X

Using the same approach as in section 4.1, we specify the observable changes that result from artefact source elimination actions. These are presented in Table 4. Note a minor adjustment required for artefact source elimination in that we consider the new execution ϖ' up until a future action X rather than directly after the occurrence of T.

Note that our representation of artefact source elimination in S-TLA reveals an additional means by which potential tampering can be identified. This is $|A_{\bar{s}}|_{\varpi} > |A_{\bar{s}}|_{\varpi'}$, which occurs if less artefacts are present in the artefact set than would typically be expected (due to the removal of observers that create those entries). This can be added to our model alongside AT and AB to further develop more features that could be used as part of Algorithm to identify occurrences of artefact source elimination and artefact destruction. As this new condition is also based on identifying an 'abnormality', we re-label AB as AB-1 and label this new condition AB-2.

We have repeated this work for the other types of tampering outlined by Harris (2006). After a careful study of the literature, we also decided to include a further type of tampering, artefact corruption. This is not explicitly included in commonly cited works on artefact tampering (typically Harris, 2006; Kessler, 2007), however, several authors have cited this as a particular type of tampering of concern, such as Saeed et al. (2020). Therefore, we have decided to include a representation of it in our model.

For the purposes of readability, we have chosen not to provide the full detail for each type of tampering in the main body of this paper. However, the result of our work provides S-TLA representations of a total of 14 types of tampering, which can be found in Appendix B.

Table 4
Observable changes to artefact set caused by artefact source elimination, along with knowledge needed to identify.

Type of tampering action (T)	Difference between ϖ and ϖ'	Knowledge required to differentiate	Real-world explanation
ICASE	$\|A_{\bar{s}}\|_{\varpi} > \|A_{\bar{s}}\|_{\varpi'}$	Whether $\|A_{\bar{s}}\|$ is within expected bounds for a given execution	Less artefacts are present than would be expected
VCASE	Elements of δ_T in the artefact set $\|A_{\bar{s}}\|_{\varpi} > \|A_{\bar{s}}\|_{\varpi'}$	Elements of δ_T Whether $\|A_{\bar{s}}\|$ is within expected bounds for a given execution	The tampering action itself leaves a trace Less artefacts are present than would be expected
IIASE	Elements of δ_X in the artefact set $\|A_{\bar{s}}\|_{\varpi} > \|A_{\bar{s}}\|_{\varpi'}$	Elements of δ_X which appear in the artefact set without the whole of δ_X Whether $\|A_{\bar{s}}\|$ is within expected bounds for a given execution	Some action can be inferred but only some of the expected traces can be found Less artefacts are present than would be expected
VIASE	Elements of δ_T in the artefact set Elements of δ_X in the artefact set	Elements of δ_T Elements of δ_X which appear in the artefact set without the whole of δ_X	The tampering action itself leaves a trace Some action can be inferred but only some of the expected traces can be found

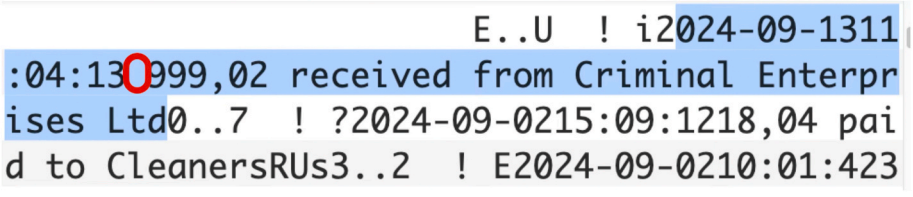

Fig. 9. Contents of partially destroyed record.

5.2. Application of the model to aid recognition of tampering

Based on the representations of tampering we have generated; we apply the same logical analysis to specify the observable change of each on a system represented in S-TLA. Unsurprisingly, several of these observable changes can be applied to more than one tampering type, as we have already seen with AB-1 and AT-1 which apply to both artefact destruction and artefact source elimination. After analysis of all the types of tampering we have represented in S-TLA, our model contains 6 such observable differences that can be used to aid the recognition tampering. These are presented, along with an informal real-world explanation of how they manifest, in Table 5.

For each, we have also developed a visual map which graphically outlines the link between each knowledge code and the types of tampering that it can be applied to recognise. The map for AB-1 and AT-1 is shown in Fig. 10 and Fig. 11 by means of an example. Images for the remaining types can be found online at https://zerotrustforensics.com,

We intend to use this model to inform the design of an automation to aid the recognition of tampering. These will be developed based on the initial version of Algorithm 1 from section 4.2. Our model allows us to reason about what tampering types can potentially be recognised by different methods. For example, we can see that a technique based solely on AB-1 and AT-2 would not be capable of identifying artefact hiding, artefact corruption or invisible complete artefact source elimination. The model can also be used to aid reasoning about the coverage of tampering provided by existing techniques of artefact tampering. For example, work by Palmbach and Breitinger (2020) proposed research aimed at recognising timestamp manipulation (a form of artefact counterfeiting) in NTFS file systems. In their work they use techniques described by AT-1 and AB-3 in Table 4. Similarly, Cho (2013) also presents a means for detecting timestamp manipulation in NTFS, using a rule-based system where the rules are constructed based on approaches AT-1 and AB-3. This work additionally has rules which seek to understand the plausibility of the data encountered, therefore also adding a way to evaluate based on AB-4. In comparison Mohamed and Khalid (2019) proposed a means detect NTFS timestamp manipulation via a machine learning approach on a simulated dataset. The authors were able to provide initial encouraging empirical results in their classification of inputs as tampered or non-tampered, achieving a classification accuracy of 95% on the synthetic dataset. However, the application of machine learning did not allow for any insight into what input features were significant for each classification. Therefore, we cannot make any statement about the reasoning used to provide each classification. This could be potentially significant if such a technique were to be applied in legal procedures, as it could not easily be shown to be based on mathematically well-founded logic.

Table 5
Full list of differences and knowledge types.

Difference between ϖ and ϖ'	Knowledge required	Label	Real-world explanation
Elements of δ_T in the artefact set	Elements of δ_T	AT-1	The tampering action itself leaves elements in the trace
Elements of δ_X in the artefact set	Elements of δ_X which appear in the artefact set without the whole of δ_X	AB-1	Some action can be inferred but only some of the expected traces can be found
$\lvert A_{\bar{s}}\rvert_{\varpi} > \lvert A_{\bar{s}}\rvert_{\varpi'}$	Whether $\lvert A\bar{s}\rvert$ is within expected bounds for a given execution	AB-2	Less artefacts are present than would be expected
Elements of δ_X not correctly formed	Whether artefacts constructed correctly in artefact set	AB-3	Artefacts have a form that indicates they were not created as part of normal operation
Elements of δ_X occur in implausible locations	How system should behave when δ_X is plausible	AB-4	The context around an action does not make logical sense or fit with normal parameters
Artefact set has been transformed by Θ	How Θ behaves such that the investigator can distinguish whether Θ has likely been applied to the artefact set	AT-2	A known obfuscation behaviour such as encryption can be seen to have occurred

6. Related work

In this paper we have used a model as a tool for making observations about the impact of tampering on a system and then used this to design a means for recognising this tampering when it occurs. We briefly compare our approach to existing methods for identifying tampering as a means of evaluation.

Marrington et al. (2011) introduce a software tool called 'Computer Activity Timeline Detection' (CAT Detect) which is designed to detect inconsistencies in the timelines generated during forensic investigations. This work is based on the formalism of Computer Profiling, introduced in more detail in Marrington et al. (2010). This formalism introduces for broad types of objects in a computer system, applications, principals, content and system, which are each represented as sets. The authors then create a tool which is used to detect inconsistencies in these properties through its application to a small case study which uses a rules-based approach. This has similarities with the approach we propose; however, the CAT Detect tool is limited to inconsistencies in only the temporal properties of digital objects. The Computer Profiling formalism used was like S-TLA and could have been used as the basis for our work. However, we found it to be less descriptive in describing concepts such as 'visibility' which we introduced in section 3. Additionally, the most recent version of the CAT Detect tool that we were able to find online was from 2016 and so would unlikely be of much assistance in a modern forensic investigation.

James and Gladyshev (2014) propose the use of signature-based methods to aid the automation of reconstructing past user activities. This is achieved using a novel action-trace update time threshold which is designed to categorise digital objects into respective update patterns to create signatures. The authors demonstrate how these can then be applied to a forensic investigation through a brief case study and make their code available for contribution online. However, this collection of 'anti-forensic' signatures has not been updated in over a decade at the time of writing and have been written for only the Windows Operating System, the version of which they used is now obsolete. This limits the ability of this approach to be used today. Additionally, the use of only a signature-based approach (comparable to the 'identifying attacks' part of our model) limits the types of tampering that can be recognised in this fashion to those that have already been identified through some other investigative means. By comparison, our model can potentially recognise known attacks (through the 'identifying abnormality' part of the model) and is provides generic observations about systems which are true regardless of the operating system and version in use.

Rekhis and Boudriga (2012a) propose a formal model using state-based logic to describe a system (S-TLA). This is used to create a 'library of attacks' which is used alongside an inference-based approach to mitigate anti-forensic attacks. This approach is therefore limited to only being applicable in scenarios where a tampering action has a corresponding entry in this library of known attacks. The authors state that maintaining such a library is a pre-requisite for digital forensic investigations. However, we have not been able to find a recent version of this which could be used to assist investigations into modern systems. Furthermore, the techniques proposed relies on the existence of a 'library of actions' which formally describes every possible action on a system. The state space of such a model would be very large for even trivial systems described in this way. By comparison, our model extends S-TLA to provide precise representations of the tampering activities themselves and then uses observations of how these activities generically change the state of systems to create a generic knowledge

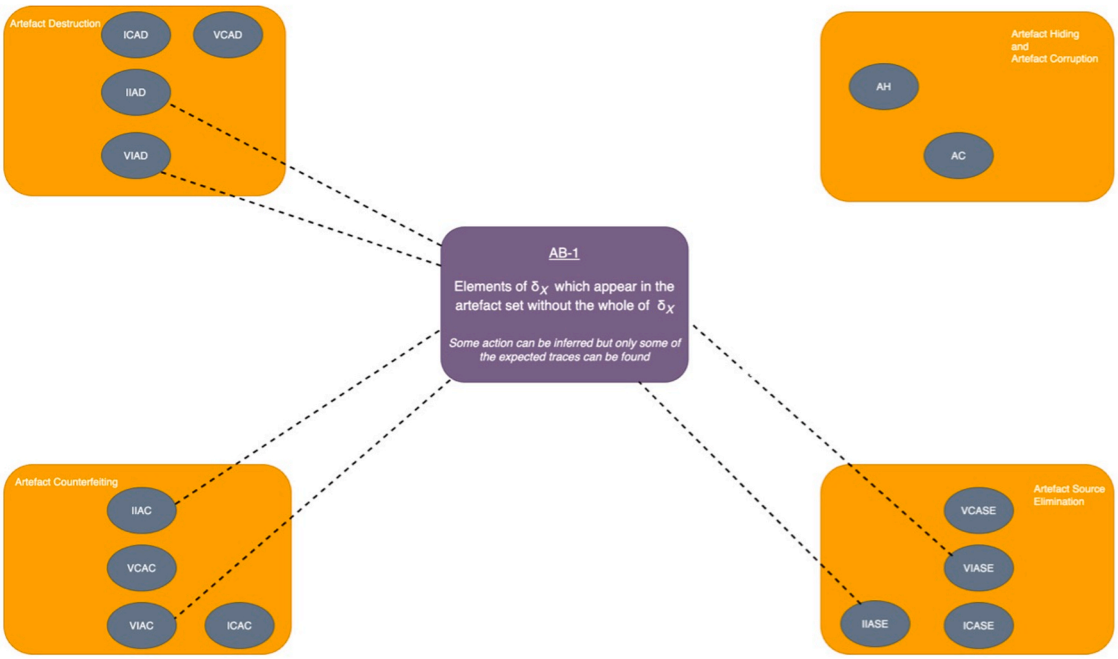

Fig. 10. Visual map of tampering that can be identified by AB-1.

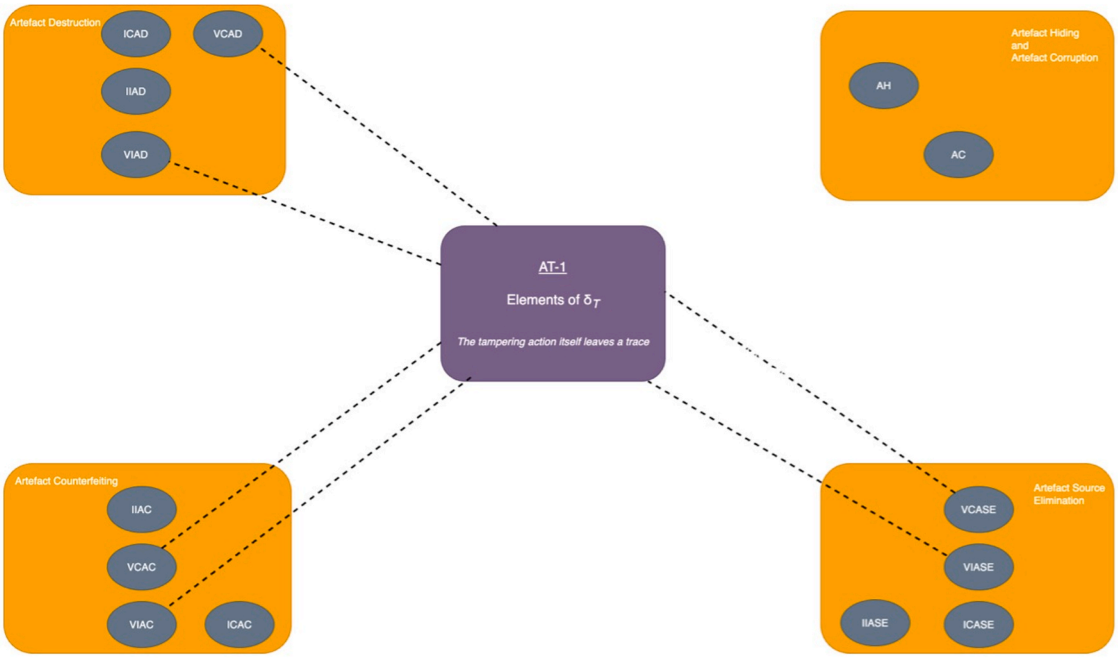

Fig. 11. Visual map of tampering that can be identified by AT-1.

framework which can be more widely applied. Our model also incorporates unknown attacks through identifying abnormality as well as already-discovered anti-forensic techniques.

Soltani and Seno (2019) propose a formal model for event reconstruction in digital forensics. Their model is based on the use of transition systems to which modal mu-calculus is applied to determine forensic properties. Once a system has been modelled, a digital forensic question is formulated and then checked against the model using the mCRL2 model checker. Using this deductive approach, the authors acknowledge the challenges presented by the explosion of the state space and show how this can be reduced using structural symmetries, i.e. machine states that are different in the model but functionally equivalent to each other. The apply their technique to the illustrative example originally presented by Gladyshev and Patel (2004) and demonstrate its utility in formally proving investigative hypotheses. However, the technique is not demonstrated for more complex real-life cases. While the authors show that structural symmetries can dramatically reduce the state-space (99% reduction is claimed) this does not eliminate the issue for more complex systems. Furthermore, this technique relies on the ability to implicitly represent the actions that the hypothesis relates to via a suitable formula. This is not done for tampering activities as it is not the focus of the author's work.

Gruber et al. (2023) also propose an alternative approach to Soltani and Seno (2019). Instead of devising new search algorithms to solve

state-space questions, the concepts of 'necessary evidence' and 'sufficient evidence' are defined and used alongside established tool for conducting model checking. The authors also use the example first proposed by Gladyshev and Patel (2004) and demonstrate their approach to solving this problem can be applied by using the NuSMV model checker. While they can demonstrate improvements to solving the problem based on existing techniques, this is a further example of a deductive approach. The authors acknowledge the challenges presented by the high time complexity of calculating the evidence sets required for their model. Additionally, the ability to generate models which are faithful to the real-life systems that are being investigated is cited as relying on human reasoning with no automation to assist in this process. Furthermore, technical limitations restricted the size of the models that were able to be processed by the authors. In comparison, our approach uses inductive logic to propose generic patterns that can be used to recognise tampering on any system without the need to create a specific model of it.

Vanini et al. (2024) propose a framework to assess the tamper resistance of data sources that are used during event reconstruction. This work is aimed at evaluating whether these data sources are resistant to potential tampering rather than evaluating whether a specific artefact may have been tampered with. A scoring system is proposed, based on factors which influence the difficulty of tampering with a digital object. This includes properties such as visibility, permissions and the existing of software that can interact with it. The authors demonstrate the importance of explicitly evaluating the possibility that digital objects have been tampered with. This is different from our work in that they do not seek to recognise specific instances of tampering, which is the purpose of our model.

7. Conclusions and future work

The ability to recognise tampering is an important part of a digital investigation. However, current descriptions of tampering are imprecise which makes recognising it challenging. Additionally, these natural language descriptions are incompatible with formal methods, which is an established approach to event reconstruction in digital forensics. Our approach provided the following contributions:

- An extension to the S-TLA model of a system to include the concept of the visibility of specific actions
- Precise representations of artefact tampering in terms of S-TLA
- A model of tampering that describes the impact of tampering actions on a system
- A knowledge framework that identifies the properties of tampering to aid recognition of it
- An algorithm that can be applied to the task of recognising tampering along with two short case studies

S-TLA is a useful framework for reasoning about events that occur on a general system. The knowledge framework provided in Table 5 provides a means by which tampering actions can be understood by investigators, independent of their technical knowledge of any particular system. This allows for the development of techniques for recognising tampering, both generally and on specific systems of interest. At this stage in our research, we have developed an initial algorithm that demonstrates the utility of the knowledge framework for the purpose of recognising tampering.

As noted in earlier sections, this algorithm is in early stages of development therefore there are some limitations to our work. Currently, the approach does not account for various factors, such as whether a feature strongly, or weakly indicates tampering (or non-tampering). This could be accounted for by introducing weighting on the features. We are choosing to take an approximation approach using inductive reasoning, rather than contend with the limitations have been already noted as applying to formal model-checking techniques when deductive approaches are used.

In future work, we plan to develop our approach further to account for more complex scenarios to demonstrate its applicability to real-life cases through empirical experimentation. This will require further development of Algorithm 1 presented in section 4.2 and validation of its applicability to representative investigative data, such as that presented by Schmitt (2018). Alongside this, there would be significant engineering effort required to understand suitable features for specific types of artefacts. One consideration is the implementation of an open-format structure, such as a template, which could be used for the development of suitable features by the wider digital forensics community.

We would also consider additional requirements that investigators may have, such as a more sophisticated means for understanding the relationships between a TAMPERED result, and the specific type of tampering that has occurred. Additionally, we would consider that future iterations of our approach would enable the algorithms to provide some means of reporting contextual information about features that were significant in the finding to provide a starting point for further investigation. Future work may also include integrating our approach into an automation which integrates with forensic toolkits and other techniques for feature extraction, such as machine learning.

CRediT authorship contribution statement

Christopher Neale: Conceptualization, Methodology, Writing – original draft. **Ian Kennedy:** Supervision, Writing – review & editing. **Bashar Nuseibeh:** Supervision, Writing – review & editing.

Declaration of interest

The authors declare that they have no known or competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. – glossary of terms in S-TLA

This appendix contains a glossary of terms and notation used in S-TLA.

Term	Notation	Explanation
System	$\mathscr{V}$	A system is a set of variables
State (system)	s	The set of values of each variable in $\mathscr{V}$ at a given moment in time
Action	$A(s_i, s_{i+1})$ (or other suitable letter – we use T as convention for a tampering action)	An action that occurs on a system. Expressed as a function from state s_i to s_{i+1}
Set of possible actions	Acts	The set of all possible actions that can occur on a system. Note that traditional model-checking techniques rely on being able to enumerate this set, whereas this is unrealistic in most cases, especially where tampering involves subverting expected types of actions

(continued on next page)

(*continued*)

Term	Notation	Explanation
System execution	ϖ	The sequence of states that occur when a system runs
Observers	O	The applications on a system which have access to a subset of system variables. Examples include operating system logging applications and security products, which may have access to a wide array of these. Additionally, typical applications will have access to the system variables related to at least their own function
Observation function	obs(s_i)	Function which returns the variables that can be observed by observers in O at state s_i. This is not a static function of the system itself, but the observers employed. For example, an operating system running in debugging mode may allow for the observation of many more system variables than under standard circumstances.
Trace	$tr(\varpi)$	The sequence of all observed variables during a system execution (i.e. the sequence that replaces every state s_i with obs(s_i))
Artefact	$a(tr(\varpi))$	The output of an observer in O taking in the trace function as an input to produce some output. For example, when an operating system notes that a new process has been launched (through variables in $tr(\varpi)$), a log is written to a log file
Artefact container	$ac(\varpi)$	The set of artefacts created by a specific observer
Artefact set	$A_{\bar{s}}$	The set of all artefacts created by all observers. Note that in the original work, this was split into the set of artefacts that could be trusted (A_s) and those that could not ($A_{\bar{s}}$). We take the position that no artefacts should be inherently trusted and so use the latter to denote the entire artefact set
Action visibility	δ_A	The subset of $A_{\bar{s}}$ which exists due to the action A. Note that this is a concept not included in the original work and has been introduced in this paper

Appendix B. – definitions of other tampering actions

This appendix contains the definitions of other types of artefact tampering using the same model as used in our paper.

Definition (ICASE): An action $T(s_i,s_{i+1})$ is invisible complete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T = \{\}$ and $\delta_X = \{\}$ for some future action X

Definition (VCASE): An action $T(s_i,s_{i+1})$ is visible complete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T \neq \{\}$ and $\delta_X = \{\}$ for some future action X

Definition (IIASE): An action $T(s_i,s_{i+1})$ is invisible incomplete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T = \{\}$ and $\delta_X \neq \{\}$ for some future action X

Definition (VIASE): An action $T(s_i,s_{i+1})$ is visible incomplete artefact source elimination if as a result:

$O \rightarrow O'$ such that $O' \subset O$, $\delta_T \neq \{\}$ and $\delta_X \neq \{\}$ for some future action X

Definition (ICAC): An action $T(s_i,s_{i+1})$ is invisible complete artefact counterfeiting if as a result:

$A_{\bar{s}} \rightarrow A'_{\bar{s}}$ such that $A_{\bar{s}} \subset A'_{\bar{s}}$, $\delta_T = \{\}$ and $\delta_X \subset A'_{\bar{s}}$ but is not a subset of $A_{\bar{s}}$ for some action $X \neq T$

Definition (VCAC): An action $T(s_i,s_{i+1})$ is visible complete artefact counterfeiting if as a result:

$A_{\bar{s}} \rightarrow A'_{\bar{s}}$ such that $A_{\bar{s}} \subset A'_{\bar{s}}$, $\delta_T \neq \{\}$ and $\delta_X \subset A'_{\bar{s}}$ but is not a subset of $A_{\bar{s}}$ for some action $X \neq T$

Definition (IIAC): An action $T(s_i,s_{i+1})$ is invisible incomplete artefact counterfeiting if as a result:

$A_{\bar{s}} \rightarrow A_{\bar{s}'}$ such that $A_{\bar{s}} \subset A'_{\bar{s}}$, $\delta_T = \{\}$ and $\delta'_X \subset A_{\bar{s}'}$ but is not a subset of $A_{\bar{s}}$ for some action $X \neq T$ where $\delta'_X \subset \delta_X$

Definition (VIAC): An action $T(s_i,s_{i+1})$ is visible incomplete artefact counterfeiting if as a result:

$A_{\bar{s}} \rightarrow A_{\bar{s}'}$ such that $A_{\bar{s}} \subset A'_{\bar{s}}$, $\delta_T \neq \{\}$ and $\delta'_X \subset A_{\bar{s}'}$ but is not a subset of $A_{\bar{s}}$ for some action $X \neq T$ where $\delta'_X \subset \delta_X$

Definition (AH): An action $T(s_i,s_{i+1})$ is artefact hiding if as a result:

$A_{\bar{s}} \rightarrow A'_{\bar{s}}$ such that $A'_{\bar{s}} \neq A_{\bar{s}}$, $\forall\, a\left(tr(\varpi)^i\right) \in A_{\bar{s}}$ such that $a\left(tr(\varpi)^i\right) \neq a'\left(tr(\varpi)^i\right)$, there exists a transformation function Θ such that $\Theta(A_{\bar{s}}) = A_{\bar{s}}$' and there exists an inverse function Θ^{-1} such that $\Theta^{-1}(A'_{\bar{s}}) = A_{\bar{s}}$.

Definition (AC): An action $T(s_i,s_{i+1})$ is artefact corruption if as a result:

$A_{\bar{s}} \rightarrow A'_{\bar{s}}$ such that $A'_{\bar{s}} \neq A_{\bar{s}}$, $\forall\, a\left(tr(\varpi)^i\right) \in A_{\bar{s}}$ such that $a\left(tr(\varpi)^i\right) \neq a'\left(tr(\varpi)^i\right)$, there exists a transformation function Θ such that $\Theta(A_{\bar{s}}) = A_{\bar{s}}$' and there does not exist an inverse function Θ^{-1} such that $\Theta^{-1}(A'_{\bar{s}}) = A_{\bar{s}}$.

Data availability

No data was used for the research described in the article.

References

Carrier, B., Spafford, E., 2004. An event-based digital forensic investigation framework. Digit. Forensic Res. Workshop 1–12. https://doi.org/10.1145/1667053.1667059.

CAS 2020/O/6689 World Anti-doping Agency V. Russian Anti-doping Agency, 2020.

Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, T., 2015. An ontology-based approach for the reconstruction and analysis of digital incidents timelines. Digit. Investig 15, 83–100. https://doi.org/10.1016/j.diin.2015.07.005.

Cho, G.S., 2013. A computer forensic method for detecting timestamp forgery in NTFS. Comput. Secur. 34, 36–46. https://doi.org/10.1016/j.cose.2012.11.003.

de Beer, R., Stander, A., Van Belle, J.P., 2015. Anti-forensics: a practitioner perspective. Int. J. Cyber-Secur. Digit. Forens. https://doi.org/10.1017/CBO9781107415324.004.

Dewald, A., 2015. Characteristic evidence, counter evidence and reconstruction problems in forensic computing. IT - inf. Tech 57, 339–346. https://doi.org/10.1515/itit-2015-0017.

Franqueira, V.N.L., Horsman, G., 2020. Towards sound forensic arguments: Structured argumentation applied to digital forensics practice. Forensic Sci. Int. Digit. Investig. 32 (300923). https://doi.org/10.1016/j.fsidi.2020.300923.

Gladyshev, P., Patel, A., 2004. Finite State Machine Approach to Digital Event Reconstruction.

Gruber, J., Humml, M., Schröder, L., Freiling, F.C., 2023. Formal Verification of Necessary and Sufficient Evidence in Forensic Event Reconstruction Keywords: Digital Evidence Digital Investigations Forensic Event Reconstruction Formal Methods Forensic Computing, In: Proceedings of Digital Forensics Research Conference Europe. DFRWS EU, 2023.

Harris, R., 2006. Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem. Digit. Investig 3, 44–49. https://doi.org/10.1016/j.diin.2006.06.005.

James, J., Gladyshev, P., Abdullah, M.T., Yuandong, Z., 2009. Analysis of Evidence Using Formal Event Reconstruction, In: International Conference on Digital Forensics and Cyber Crime.

James, J.I., Gladyshev, P., 2014. Automated inference of past action instances in digital investigations. https://doi.org/10.1007/s10207-014-0249-6.

Kessler, G.C., 2007. Anti-forensics and the digital investigator. Int. J. Comput. Sci. Imformation Technol. IJCSIT 3, 1–7. https://doi.org/10.4225/75/57ad39ee7ff25.

Lamport, L., 1999. Specifying concurrent systems with TLA+ 20–27. https://doi.org/10.1145/75200.75203.

Margot, P., 2011. Forensic science on trial - what is the law of the land? Aust. J. Forensic Sci. 43, 89–103. https://doi.org/10.1080/00450618.2011.555418.

Marrington, A., Baggili, I., Mohay, G., Clark, A., 2011. CAT detect (computer activity timeline detection): a tool for detecting inconsistency in computer activity timelines. Digit. Investig 8, S52–S61. https://doi.org/10.1016/j.diin.2011.05.007.

Marrington, A., Mohay, G., Morarji, H., Clark, A., 2010. A model for computer profiling. In: Proceedings of the 5th International Conference on Availability, Reliability, and Security. IEEE, pp. 15–18.

Michelet, G., Schneider, J., Withanage, A., Breitinger, F., 2026. Hey GPT-OSS, looks like you got it – now walk me through it! an assessment of the reasoning language models chain of thought process for digital forensics. Forensic Sci. Int. Digit. Investig. 56 (302052). https://doi.org/10.1016/j.fsidi.2026.302052.

Mohamed, L.A., Khalid, C., 2019. Detection of timestamps tampering in ntfs using machine. Procedia Comput. Sci. 160, 778–784. https://doi.org/10.1016/j.procs.2019.11.011.

Neale, C., 2023. Fool me once: a systematic review of techniques to authenticate digital artefacts. Forensic Sci. Int. Digit. Investig. 45 (301516). https://doi.org/10.1016/j.fsidi.2023.301516.

Neale, C., Kennedy, I., Price, B., Yu, Y., Nuseibeh, B., 2022. Forensic science international : Digital investigation the case for zero trust digital forensics. Forensic Sci. Int. Digit. Investig 40 (301352). https://doi.org/10.1016/j.fsidi.2022.301352.

Nemetz, S., Schmitt, S., Freiling, F., 2018. A standardized corpus for SQLite database forensics. DFRWS 2018 EU - Proc. 5th Annu. DFRWS eur, 24, S121–S130. https://doi.org/10.1016/j.diin.2018.01.015.

OpenText Corporation, n.d. EnCase.

Palmbach, D., Breitinger, F., 2020. Forensic science international : Digital investigation artifacts for detecting timestamp manipulation in NTFS on windows and their reliability. Forensic Sci. Int. Digit. Investig. 32 (300920). https://doi.org/10.1016/j.fsidi.2020.300920.

Pollitt, M., Casey, E., Jaquet-Chiffelle, D.-O., Gladyshev, P., 2018. A framework for harmonizing forensic science practices and Digital/Multimedia evidence. Organ. Sci. Area comm. Forensic Sci. OSAC 1–29.

Rekhis, S., Boudriga, N., 2010. Formal digital investigation of anti-forensic attacks. 5th Int. Workshop Syst. Approaches Digit. Forensic Eng. https://doi.org/10.1109/SADFE.2010.9. SADFE 2010 33–44.

Rekhis, S., Boudriga, N., 2012a. A system for formal digital forensic investigation aware of anti-forensic attacks. IEEE Trans. Inf. Forensics Secur. 7, 635–650. https://doi.org/10.1109/TIFS.2011.2176117.

Rekhis, S., Boudriga, N., 2012b. A hierarchical visibility theory for formal digital investigation of anti-forensic attacks. Comput. Secur. 31, 967–982. https://doi.org/10.1016/j.cose.2012.06.009.

Saeed, S.H., Arash, H.L., Ghorbani, A.A., 2020. Forensic science international : digital investigation A survey and research challenges of anti-forensics : evaluation of game-theoretic models in simulation of forensic agents ' behaviour * 35. https://doi.org/10.1016/j.fsidi.2020.301024.

Schmitt, S., 2018. 2018. Introducing anti-forensics to SQLite corpora and tool testing. Proc. - 11th int. Conf. IT secur. Incid. Manag. IT forensics IMF, 89–106. https://doi.org/10.1109/IMF.2018.00014.

Schneider, J., Düsel, L., Lorch, B., Drafz, J., Freiling, F., 2022. Forensic Science international : Digital investigation Prudent design principles for digital tampering experiments. Forensic Sci. Int. Digit. Investig 40 (301334). https://doi.org/10.1016/j.fsidi.2022.301334.

Soltani, S., Seno, S.A.H., 2019. A formal model for event reconstruction in digital forensic investigation. Digit. Investig 30, 148–160. https://doi.org/10.1016/j.diin.2019.07.006.

Vanini, C., Gruber, J., Hargreaves, C., Benenson, Z., Freiling, F., Breitinger, F., 2024. Strategies and Challenges of Timestamp Tampering for Improved Digital Forensic Event Reconstruction (Extended Version).

Wieringa, R., 2014. Design Science Methodology for Information Systems and Software Engineering. Springer International Publishing.

Willassen, S.Y., 2008. Methods for Enhancement of Timestamp Evidence in Digital Investigations, Phd Thesis.

X-Ways Software Technology, n.d. X-Ways.

Zdzichowski, P., Sadlon, M., Väisänen, T.U., 2015. Anti-Forensic Study.