Online Appendix to: Forensic Analysis of Database Tampering

KYRIACOS E. PAVLOU and RICHARD T. SNODGRASS
University of Arizona

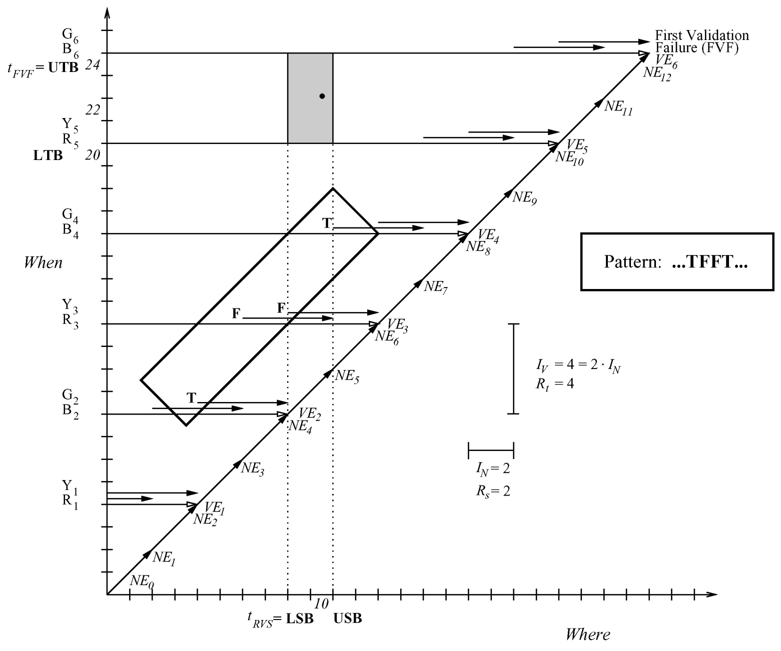
This appendix has six sections. Appendix A discusses the subtleties involved in the forensic analysis of introactive corruption events, while Appendix B demonstrates how false positives arise in the RGBY algorithm. Appendix C describes the Tiled Bitmap algorithm (pseudocode provided), discusses the notion of a candidate set, and gives the running time of the algorithm. A more thorough exposition of the use of candidate sets in forensic analysis may be found elsewhere [Pavlou and Snodgrass 2006b]. The remaining Appendices D, E, and F provide the forensic cost for the algorithms, using worst-case, best-case, and average-case assumptions, respectively, on the distribution of corruption sites.

A. INTROACTIVE CORRUPTION EVENTS

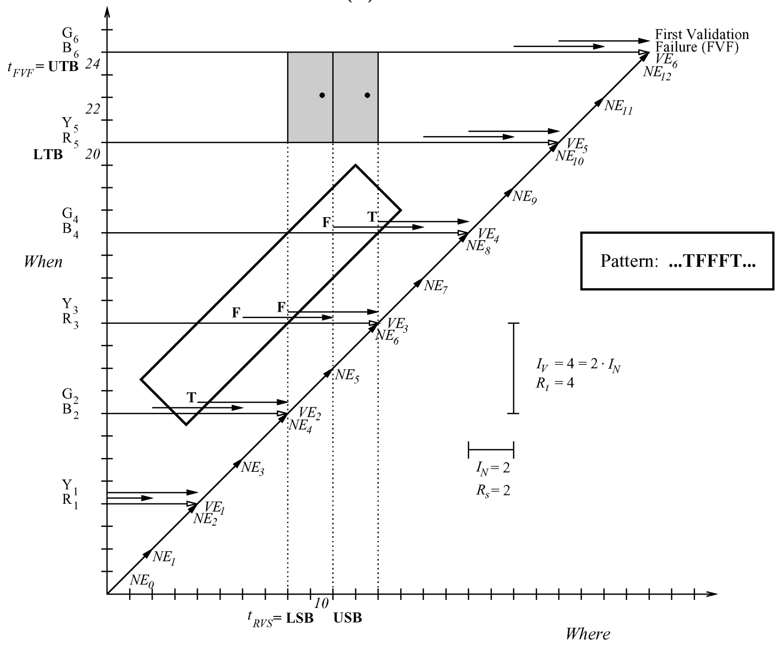
Introactive corruption events were introduced in Section 5. However, subsequent examples and algorithms do not deal explicitly with the particular challenges raised by these CEs in forensic analysis. The main challenge stems from the fact that the partial chains computed during the validation event scan terminating at t_{FVF} cannot be used to identify introactive CEs (this holds for all algorithms utilizing partial hash chains). The reason is that an introactive CE occurs *before* these latest partial hash chains are notarized. Recall that we deferred the partial chain hashing and notarization during a validation scan in order to decrease the read overhead. This results in the latest partial chains hashing the corrupted values. Hence the validation of the rehashed value corresponding to the entire database must happen first, and if and only if it returns true are the partial hash chains notarized. Moreover, because the cumulative black chains perform hashing in real-time, it is impossible for an introactive CE to occur before their creation. This implies that a single introactive CE *can* be detected by all algorithms because, in this case, we can locate the corruption using only the cumulative black chains. This problem only arises if there are multiple CEs, as in the example shown in Figure 7. In this example, partial chains B_6 and G_6 cannot be trusted. The analysis and implementation of the algorithms do not deal with this explicitly. The working assumption in the presentation of the algorithms in this article is that all partial chains can be used in forensic analysis. One way to accommodate introactive CEs is for each algorithm to treat the entire region where introactive CEs can occur as suspect when dealing with multiple corruptions.

B. FALSE POSITIVES IN THE RGBY ALGORITHM

In this section, we discuss the nature of the linear search of the RGBY Algorithm and show how false positives inevitably arise. Figure 18(a) shows the

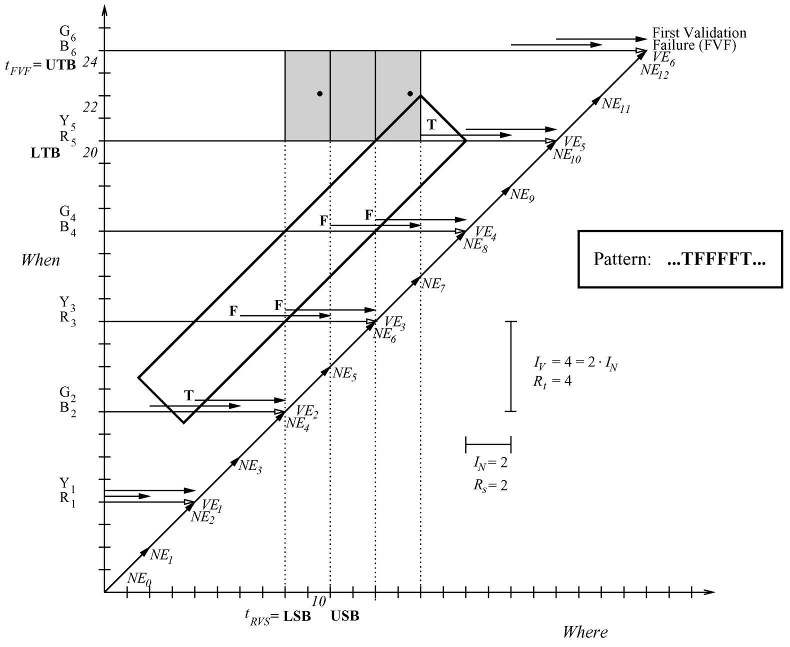


(a) $\kappa = 1$

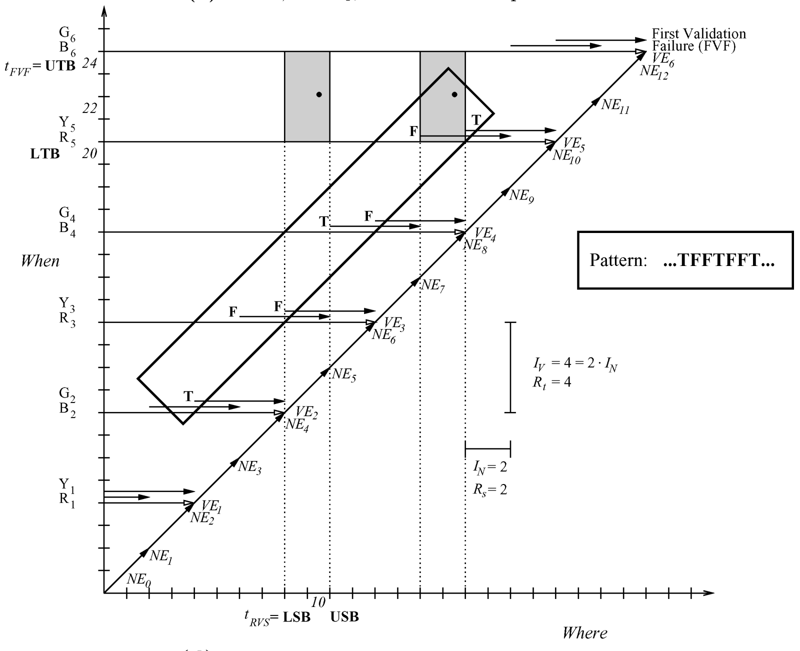


(b) $\kappa = 2$; contiguous corruption sites

Fig. 18. The chain patterns and corresponding corruption regions of the RGBY algorithm for one or two contiguous corruption sites.



(c) $\kappa = 2$; one I_N between corruption sites



(d) $\kappa = 2$; two I_Ns between corruption sites

Fig. 18. (*continued*) The chain patterns and corresponding corruption regions of the RGBY algorithm for two corruption sites with one or two I_Ns between the sites. The middle corruption region rectangle in (c) is a false positive.

basic pattern of truth values encountered over a single corruption during the linear scan of the forensic analysis, namely ...TFFT... Similarly, if we look at Figure 18(d) we observe that two corruption sites sufficiently-spaced (i.e., distance two or more I_N apart in the spatial dimension) produce a succession of the same basic pattern observed in the case where $\kappa = 1$, that is, ...TFFTFFT... In other words, sufficiently spaced multiple corruption sites can be definitively identified by the RGBY Algorithm just by seeking the pattern ...FF... during the linear scan.

If however, two corruption sites are less than two I_N apart (in the spatial dimension) then the situation is more complex. Figure 18(b) depicts two *contiguous* corruption sites. The pattern observed here is ...TFFFT... It is important to realize that since the linear scan involves a look-ahead of size one (i.e., it needs to examine the results of two chains at a time to locate ...FF...) and between each iteration the frame shift is again one, the pattern ...FFF... will be correctly interpreted by the algorithm as two ...FF... patterns overlapping in the middle, hence correctly identifying the two contiguous corruption sites.

This does not happen in the the case shown in Figure 18(c) where two corruption sites are distance I_N apart. Here, the pattern observed is ...TFFFFT... Parsing this string as before, that is, two values at a time, will result in the algorithm identifying *three contiguous* corruption sites instead of the correct two. Thus the corruption is overestimated and the middle $I_N \times I_V$ rectangle is a false positive. Any attempt to circumvent this problem by increasing the look-ahead to three chains (i.e. parse four values at a time) is doomed because the case where there are indeed three contiguous corruption sites produces the same pattern as the case shown in Figure 18(c), making the two indistinguishable. For this reason, occurrence of false positives is inevitable in the RGBY algorithm, and moreover, in the worst-case scenario where corruption sites alternate with corruption-free areas of width I_N, RGBY can produce up to 50% false positives.

C. THE TILED BITMAP ALGORITHM

Here we present an improved version of the Polychromatic Algorithm [Pavlou and Snodgrass 2006a] called the Tiled Bitmap Algorithm. The original Polychromatic Algorithm utilized multiple *Red* and *Blue* chains while retaining the *Green* chain from the RGB Algorithm. These two kinds of chains and their asymmetry complicated this algorithm. The Tiled Bitmap Algorithm relocates these chains to be more symmetric, resulting in a simpler pattern.

We proceed to modify the Polychromatic Algorithm by:

—Removing the *Green* chain altogether.

—Adding two new subchains in the *Red* and *Blue* chain groups. For odd i, the algorithm computes the main red hash chain from $NE_{2\cdot i-3}$ to $NE_{2\cdot i-1}$, while for even i the blue and green chains are computed over the intervals $NE_{2\cdot i-3}$ to $NE_{2\cdot i-1}$ and $NE_{2\cdot i-2}$ to $NE_{2\cdot i}$, respectively.

—Shifting the first group of Red_1 to the right.

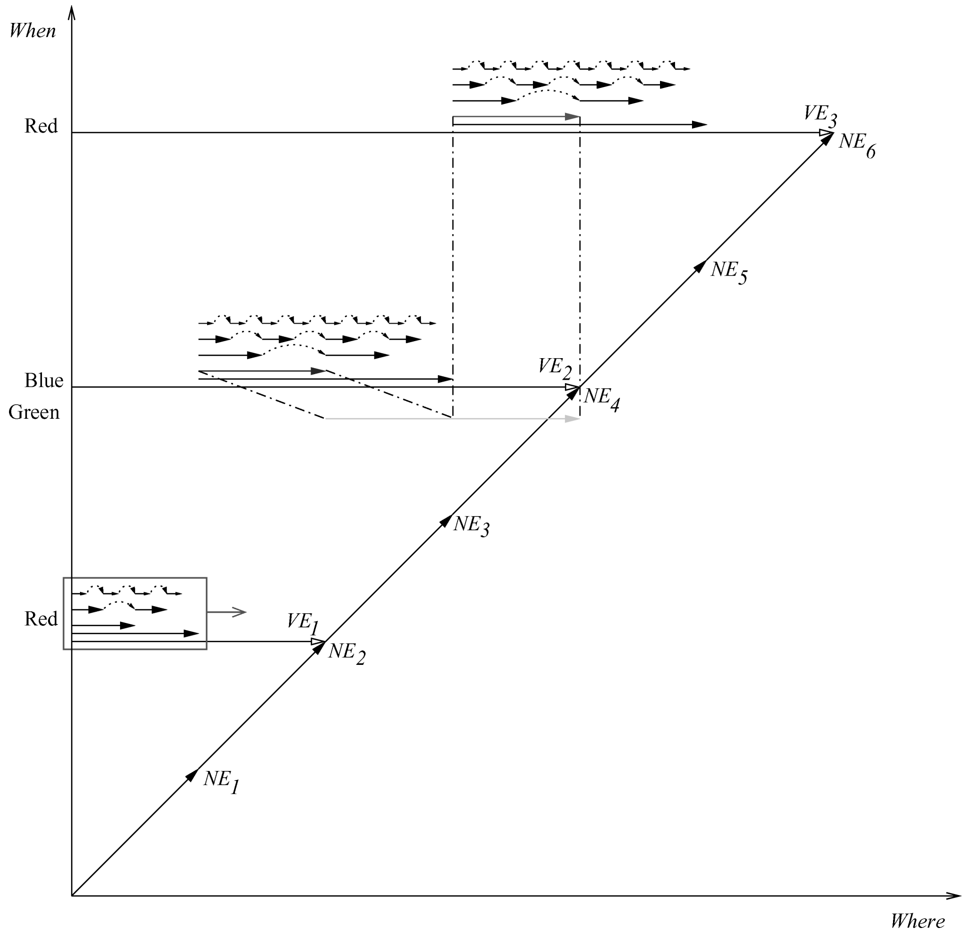


Fig. 19. Improvements introduced to the Polychromatic Algorithm.

As Figure 19 shows, the main green chain $Green_i^0$ is "broken" in half and is substituted by Red_{i+1}^1 and $Blue_i^1$. The second half of the $Green_i^0$ chain becomes the "missing" Red_{i+1}^1 chain in the next group of red chains, in order to complete the logarithmic number of chains defined in an I_V. The first half of the $Green_i^0$ chain, however, covers the complimentary interval of what is missing in the group of blue chains. Hence we first take the complement of this first half and then add it as the missing $Blue_i^1$ chain. This leads to an increase in the number of hash chains by one per I_V. This is the price paid in order to have each group of chains function as a bitmap and to have the ability to perform the combinatorial bit pattern analysis described shortly.

Next, in a completely independent step, we shift to the right the Red_1 group of hash chains (which was shorter than the rest). In this way, all the remaining hash chain groups are also shifted to the right by $I_V/2$ days. This has the result of aligning the hash chain groups, with the actual validation intervals, and thus making the structure of the algorithm more regular. Each hash chain group is repeated and thus "tiles" the action line and will serve later as a bitmap. Hence the name of this algorithm: *Tiled Bitmap Algorithm*.

Finally, we make this algorithm more general by fixing the length of the tile to cover an I_N and have V number of tiles between successive validations as shown in Figure 9 in Section 8.3.

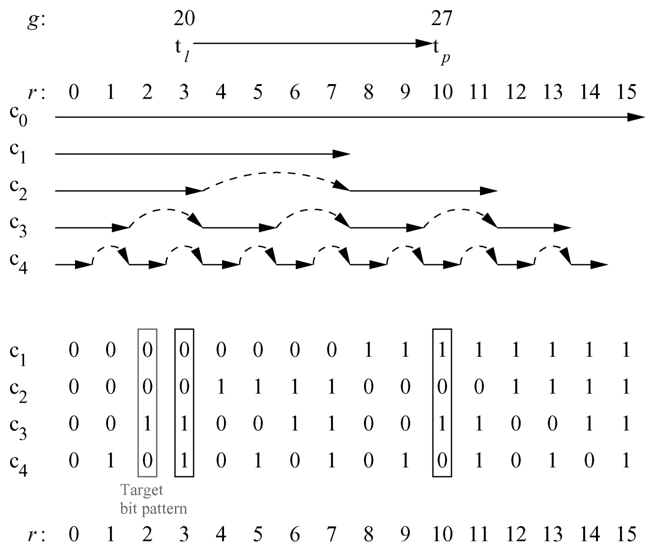
If the CE is data-only, the result of validating the entire tile of hash chains (marked with a "$\rightsquigarrow$" in Figure 9) and concatenating the result of each subchain creates a binary string whose numerical value is the relative position of the compromised granule within the tile. In this way, we can easily establish a mapping between the binary string representation of the truth value pattern (1 = Success, 0 = Failure) within each hash chain group and the desired time (granule) down to R_s.

Let us turn to an example involving a corruption. Consider CE_1 in Figure 9. We find the first tile in which a corruption has occurred via binary search in order to locate t_{RVS}. In this figure, CE_1 has $t_l = 19$ and a relative position within the second I_N of 2. If we validate the hash chains of the tile in which the CE transpired, then we get the string 00010 (most significant bit corresponds to the chain that covers all the days in I_N), termed the *target bit pattern*. The numerical value of the target string 00010 is 2, which is exactly the relative position of the granule within the second I_N.

Now, let's see what happens if a timestamp corruption occurs and both t_l and t_p are within the same tile. Figure 9 also shows a postdating CE_2 with $t_l = 20$ and $t_p = 27$, which are both in the second tile ($I_N = 16$). If each of these were to appear on their own, the target bit patterns produced by the tile validation would be 0011 (3rd granule within N) and 1010 (10th granule within N). However, since both occur at the same time within the same I_N, and the hash chains are linked together, then the bit patterns given are *AND*ed and the target 0010 is the actual result of the validation, as shown in Figure 20. This target corresponds to the existence of the two suspect days t_l and t_p, without being able to distinguish between the two. (NB: if g is a specific granule while r is its relative position within I_N, then $(g - 1) \bmod N = r$.)

In reality, the situation is more involved: when dealing with multiple CEs, there might be many combinations of bit patterns which, when *AND*ed, can yield the target bit pattern computed during forensic analysis. Thus even the simple case where a single post/backdating CE does not have its endpoints in different tiles can introduce ambiguity. For example, we cannot distinguish between the two scenarios shown in Figure 21 because in both cases the target bit pattern is the same. In the first case, both CE_2 and CE_3 produce the target bit pattern 0010 because the *AND* operation is commutative: $0011 \wedge 1010 = 1010 \wedge 0011$. For this reason, we cannot distinguish between CE_2 and CE_3. Moreover, distinguishing between CE_2, CE_3 and CE_4, CE_5 is also impossible because CE_4 and CE_5 also produce the same target bit pattern as before. More specifically, CE_4 produces the bit pattern $0010 \wedge 0111 = 0010$, and CE_5 produces again $0110 \wedge 1010 = 0010$.

Hence we introduce the notion of a candidate set [Pavlou and Snodgrass 2006b], because the pre-image of the target bit pattern under the bit-wise *AND* function is not unique. More formally, we define the length l of a binary number b, denoted by $|b| = l$, as the number of its digits. We seek to find the pre-images



r	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
c_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
c_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
c_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
c_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Fig. 20. The bitmap of a single tile.

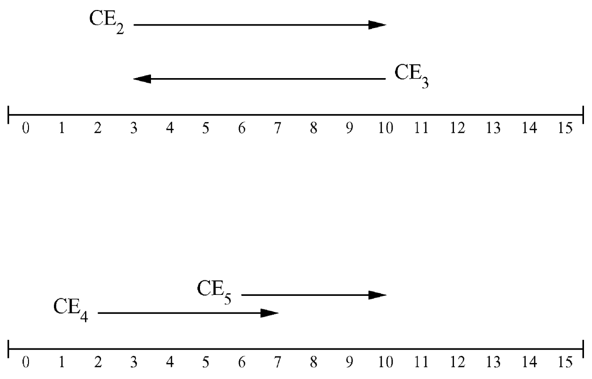


Fig. 21. Examples of CEs resulting in the same target bit pattern.

of all the binary numbers of length l, $\mathbb{B} = \{b : |b| = l\}$, under a family of bit-wise *AND* functions whose domain is a finite Cartesian product:

$$AND_k : \mathbb{B}^k \longrightarrow \mathbb{B}$$
$$AND_k((b_1, b_2, \ldots, b_k)) = b_1 \wedge b_2 \wedge \ldots \wedge b_k.$$

Observe that the maximum number k of sets participating in the Cartesian product is 2^l, since if k is allowed to take a value beyond that, it will force a repetition of one of the binary numbers. This is not informative or useful in any way, since repeated *AND*ing operations with the same binary number leave the

```
// input:   $t_{FVF}$ is the time of first validation failure
//          $I_N$ is the notarization interval
//          $V$ is the validation factor
//          $k$ is a parameter of the $C_{set}$ to be created
// output:  $C_{set}$, an array of binary numbers
//          $UTB$, $LTB$ are the temporal bounds on $t_c$
procedure Tiled_Bitmap($t_{FVF}$, $I_N$, $V$, $k$ ):
1:      $I_V \leftarrow V \cdot I_N$
2:      $t_{RVS} \leftarrow$ find_$t_{RVS}(t_{FVF}, I_N)$
3:      $UTB \leftarrow t_{FVF}$
4:      $LTB \leftarrow \max(t_{FVF} - I_V,\ t_{RVS})$
5:      $target \leftarrow 0$
6:      $C_{set} \leftarrow C_{temp} \leftarrow \emptyset$
7:      if $t_{RVS} \bmod I_V = 0$ then $t \leftarrow t_{RVS}$     // $t$ must coincide with the start of a tile
8:      else $t \leftarrow t_{RVS} - I_N$
9:      while $t < t_{FVF}$ do
10:         if $\neg$ val_check($c_0(t)$) then
11:             $n \leftarrow \lg I_V$
12:             for $i \leftarrow n$ to 1 do
13:                 $target \leftarrow target + 2^{n-i} \cdot$val_check($c_i(t)$)
14:             $C_{temp} \leftarrow$ candidateSet($target$, $n$, $k$)
15:             for each $r \in C_{temp}$ do
16:                 $g \leftarrow (r \cdot t)/I_V + 1$
17:                 $C_{set} \leftarrow C_{set} \cup \{g\}$
18:         $t \leftarrow t + I_V$
19:     return $C_{set}$, $LTB < t_c \leq UTB$
```

Fig. 22. The Tiled Bitmap Algorithm.

result invariant (the operation is *idempotent*). In other words, repetition is not allowed and hence, for a given k-tuple, all its components are distinct. Also note that the value of k uniquely identifies a specific AND_k function in this family. The *candidate set* is the set of all binary numbers that appear as components in at least one of the pre-images (i.e., k-tuples) of a specific binary number termed the *target*:

$$C_{target,k} = \{b \in \mathbb{B} \mid \exists\, b_1, b_2, \ldots, b_{k-1} \in \mathbb{B}\ (AND_k((b, b_1, \ldots, b_{k-1})) = target)\}.$$

This candidate set captures all potential sites of corruption. In this example the candidate set obtained for CE$_2$, CE$_3$ and CE$_4$, CE$_5$ will be the same in both cases and is equal to

$$C_{0010,2} = \{0010, 0011, 0110, 0111, 1010, 1011, 1110, 1111\}\ .$$

The pseudocode for the Tiled Bitmap Algorithm is provided in Figure 22. In this algorithm, the partial hash chains within a tile are denoted by $c_0(t), c_1(t), \ldots, c_{\lg N}(t)$, with $c_i(t)$ denoting the i^{th} hash chain of the tile that starts at time instant t. The algorithm begins looking at the black chains as the Polychromatic Algorithm does. This bounds t_c: $LTB < t_c \leq UTB$ as before. The binary search on the black chains also finds the value of t_{RVS}. Lines 7 and 8 adjust the start of the iteration to coincide with the beginning of a tile. On line 9 the algorithm iterates through the different tiles and checks (line 10) if the longest partial chain $c_0(t)$ evaluates to false. If not, it moves on to the next tile. If the chain evaluates to false (line 10), the algorithm iterates through the rest of the partial chains in the tile (line 12) and concatenates the result of each validation to form the target number (line 13). Then, the candidateSet function is called to compute all the pre-images of the target number according to the user-specified parameter k, discussed previously, and in more detail elsewhere [Pavlou and Snodgrass 2006b].

On lines 15–16 the candidate granules are renumbered to reflect their global position. The call to find_t_{RVS} takes $2 \cdot \lg(D/N)$ time because it performs a binary search on the cumulative black hash chains in order to locate t_{RVS}. The "while" loop on line 9 takes $\lceil D/N \rceil$ in the worst case. In reality, because of the "if" statement on line 10, the body of the loop gets executed only if corruption is initially detected by using $c_0(t)$. Hence the actual running time of the loop is $\Theta(F)$, where F is the number of times the validation of a $c_0(t)$ chain fails. The "for" loop on line 12 takes $\lg(I_N/R_s)$, while the candidateSet function takes $\Omega(\lg(I_N/R_s) + 2^z)$. The loop on line 15 takes $\Theta(2^z)$, where z is the number of zeros in the target binary number. Hence the run time of this algorithm is as follows:

$$\begin{aligned}
&\Omega(\lg(D/N) + F \cdot (\lg(I_N/R_s) + (\lg(I_N/R_s) + 2^z) + 2^z)) \\
&\quad = \Omega(\lg(D/N) + F \cdot (\lg N + 2^z)) \\
&\quad = O(\lg(D/N) + (D/N) \cdot (\lg N + N)) \\
&\quad = O(D).
\end{aligned}$$

The upper bound is obtained as follows. F in the worst case is $O(D/N)$, that is, the total number of tiles. 2^z in the worst case is N because that is the total number of granules (R_s units) within a tile.

D. FORENSIC COST FOR WORST-CASE DISTRIBUTION OF CORRUPTION SITES

In Section 9.1, we analyzed the worst-case forensic cost for the Monochromatic Algorithm. Here, we proceed with a similar analysis for the RGBY, Tiled Bitmap, and a3D forensic analysis algorithms.

D.1 The RGBY Algorithm

As with the previous algorithm, in the RGBY Algorithm, the spatial detection resolution (R_s) is I_N, so after normalizing by R_s, $N = 1$. Also recall that $V = 2$ for this algorithm. In this algorithm, during normal processing at each validation event, we validate one chain and notarize two partial chains; hence we have $(D/2) \cdot 3$ interactions with the digital notarization service during validation.

During forensic analysis, we have to perform a linear search, which could involve all partial hash chains previously notarized, that is, two at each validation event, and hence $2 \cdot (D/2)$.

$$\begin{aligned}
\mathit{NormalProcessing}_{RGBY} &= \mathit{Number\ of\ Notarizations} \\
&\quad + \mathit{Number\ of\ Validations} \\
&= D \\
&\quad + 3 \cdot (D/2)
\end{aligned}$$

Table XI. Forensic Areas for $1 \leq \kappa \leq D$ Corruption Sites (RGBY)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	2	0	$TotalArea - 2$
2	4	0	$TotalArea - 4$
3	4	0	$TotalArea - 4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
κ	4	0	$TotalArea - 4$

$$\begin{aligned} ForensicAnalysis_{RGBY} &= \textit{Binary search for finding } t_{RVS} \\ &\quad + \textit{Linear scan of partial chains} \\ &= 2 \cdot \lg(D) \\ &\quad + 2 \cdot (D/2) \end{aligned}$$

The RGBY Algorithm can detect multiple corruption sites which, if sufficiently apart, can produce distinct $Area_P$, each equal to $V \cdot N^2 = 2$. In the worst case, however, if corruptions alternate with corruption-free areas of spatial dimension I_N, then the RGBY algorithm produces false positives by identifying the intervening corruption-free area as part of $Area_P$, as shown in Figure 23. This makes $Area_P = 4$ for all $\kappa > 1$, and $Area_U = 0$.

$$\begin{aligned} FC_{RGBY}(D, 1, 2, \kappa) &= (D + 3 \cdot (D/2) + 2 \cdot \lg D + 2 \cdot (D/2)) \\ &\quad + \left(2 + \sum_{i=2}^{\kappa} 4\right) \end{aligned}$$

For $\kappa = 1$, the last term is an empty sum, and thus is equal to zero. Hence:

$$\begin{aligned} FC_{RGBY}(D, 1, 2, 1) &= (D + 3 \cdot (D/2) + 2 \cdot (D/2) + 2 \cdot \lg D) + 2 \\ &= O(D) \\ FC_{RGBY}(D, 1, 2, \kappa \geq 2) &= (D + 3 \cdot (D/2) + 2 \cdot (D/2) + 2 \cdot \lg D) \\ &\quad + (2 + (\kappa - 1) \cdot 4) \qquad (3) \\ &= O(\kappa + D). \end{aligned}$$

D.2 The Tiled Bitmap Algorithm

Unlike the previous two algorithms, the Tiled Bitmap Algorithm effects a spatial resolution (R_s) that is *smaller* than the notarization interval. Also V and N are both set by the DBA, and hence appear as variables.

The normal processing component is made up of the number of notarizations required for the black chains, the number of notarizations for the partial chains that make up each tile, and the number of validations performed:

$$\begin{aligned} NormalProcessing_{tiled\_bitmap} &= \textit{Number of black chain notarizations} \\ &\quad + \textit{Number of within-tile notarizations} \\ &\quad + \textit{Number of validations.} \end{aligned}$$

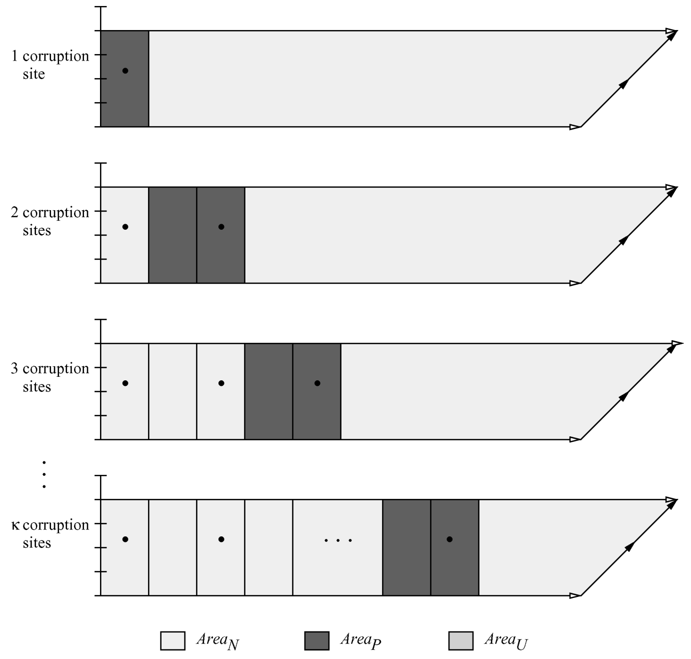


Fig. 23. Three types of forensic area for RGBY and κ corruption sites.

For each validation event, during normal processing, the Tiled Bitmap Algorithm contacts the notarization service once to validate the database and then notarizes all the chains within a tile, which are $1 + \lg N$ in number:

$$\begin{aligned} \mathit{NormalProcessing}_{\mathit{tiled\_bitmap}} &= D/N \\ &\quad + (1 + \lg N) \cdot (D/N) \\ &\quad + D/(V \cdot N). \end{aligned}$$

For forensic analysis, we have to perform a binary search to find t_{RVS} and then a linear search to locate corruptions for each tile. The linear search could involve κ tiles in the worst case:

$$\begin{aligned} \mathit{ForensicAnalysis}_{\mathit{tiled\_bitmap}} &= \mathit{Binary\ search\ for\ finding\ } t_{RVS} \\ &\quad + \mathit{Number\ of\ chains\ validated\ within\ tiles} \\ &= 2 \cdot \lg(D/N) \\ &\quad + (1 + \lg N) \cdot \kappa. \end{aligned}$$

The algorithm returns a candidate set $C_{target,2}$, each element of which corresponds to a distinct corruption region of area $V \cdot N^2$. The cardinality of the candidate set is equal to 2 raised to the number of zeros z in the target [Pavlou and Snodgrass 2006b] and thus $|C_{target,2}| = 2^z$. The maximum value z can take is the length l of the *target*, which is $\lg N$. This implies that $|C_{target,2}| = O(2^{\lg N}) = O(N)$, as it should be (!).

Note that the worst-case scenario for the Tiled Bitmap Algorithm occurs when each of the κ corruption sites occurs in the first granule of each tile as

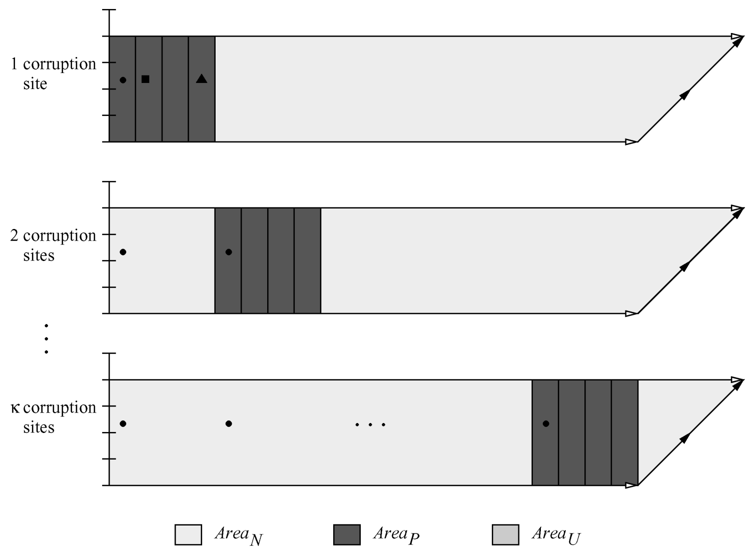


Fig. 24. Three types of forensic area for Tiled Bitmap and κ corruption sites.

Table XII. Forensic Areas for $1 \le \kappa \le D$ Corruption Sites (Tiled Bitmap)

# Corruption Sites $(1 \le \kappa \le D)$	$Area_P$	$Area_U$	$Area_N$
1	$V \cdot N^2$	0	$TotalArea - V \cdot N^2$
2	$V \cdot N^2$	0	$TotalArea - V \cdot N^2$
3	$V \cdot N^2$	0	$TotalArea - V \cdot N^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
κ	$V \cdot N^2$	0	$TotalArea - V \cdot N^2$

shown with a • in Figure 24. Subsequent corruption sites (shown with ■ and ▲) within the same tile as the ones in the first tile do not alter the cardinality of the candidate set and thus do not cause an increase in $Area_P$. Note that $Area_P$ is the entire tile in this (improbable) worst case. We still normalize I_N and I_V by R_s, which implies that N is larger than 1 (actually, it must be a power of 2):

$$\begin{aligned} FC_{tiled\_bitmap}(D,N,V,\kappa) &= ((D/N) + (1+\lg N)\cdot(D/N) + D/(V\cdot N) \\ &\qquad + 2\cdot\lg(D/N) + (1+\lg N)\cdot\kappa) \\ &\quad + (\kappa\cdot V\cdot N^2) \qquad (4) \\ &= O(\kappa\cdot V\cdot N^2 + (D\cdot\lg N)/N + \lg D). \end{aligned}$$

In this case, even though the cost of normal processing and forensic analysis have increased because of the increased number of notarizations and validations that need to be performed, the area has shrunk considerably. The entire $Area_U$ is zero, while $Area_P$ has seen a modest increase.

It is worth noting here that there exists a case when the candidate set will find the corruption site with perfect precision. This happens when the corruption only occurs inside the last granule of the tile (shown with ▲ and disregarding the other corruption sites in that tile). In this case, the resulting *target* bit string uniquely identifies the corrupted granule so we know that there exists only one corruption site in the tile, along with its exact location.

D.3 The a3D Algorithm

In the a3D Algorithm, during normal processing, for every validation event, we notarize one cumulative black chain, we validate once the entire database, and we notarize a number of partial hash chains depending on the R_s unit. The total number of notarizations performed in D units was calculated in Section 8.4; see equation (1) in that section. (Recall that we normalize I_N and t_{FVF} by R_s, the spatial detection resolution. Recall also that in the a3D Algorithm, $V = 1$.) There we proved that the total number of notarizations is equal to $O(D)$:

$$\begin{aligned} \mathit{NormalProcessing}_{a3D} &= \mathit{Total\ Number\ of\ Validations} \\ &\quad + \mathit{Total\ Number\ of\ Notarizations} \\ &= D/N \\ &\quad + \mathcal{N}(D) + D/N - (1 + \lfloor \lg(D/N) \rfloor) \\ &= O(D). \end{aligned}$$

The forensic analysis cost depends on the actual distribution of the κ corruption sites. A worst-case scenario arises when each successive corruption site that is added causes the maximum possible number of validations in the algorithm. To explain this, we utilize the binary tree representation of the hash chains in the algorithm. The algorithm is forced to perform the maximum number of validations whenever a new site corrupts a leaf that belongs to a subtree rooted at a node whose previous validation has yielded a true result, and this subtree has maximal height. Figure 25 shows a tree of height four and the validation results after the addition of $\kappa = 8$ corruptions sites. A (nonunique) sequence of adding corruption sites, which satisfies the condition for worst-case scenario stated previously, is given with numbers underneath the leaves. For example, the first site in Figure 25 is a corruption on the data covered by hash chain $P_{1,0,0}$, the second site corrupts data covered by hash chain $P_{5,0,8}$, and so on. It's easy to see that the existence of $\kappa = D/2$ properly distributed corruption sites can force the validation of all the hash chains covering the first D units.

The number of validations in forensic analysis with each successive addition of a corruption site, satisfies the following recursive formula:

$$\mathcal{V}(\kappa) = \mathcal{V}(\kappa - 1) + 2 \cdot (H - depth(\kappa)), \quad (5)$$

where $\mathcal{V}(\kappa)$ is the number of hash chains validated by the algorithm when κ corruption sites exist under a worst-case distribution, the height of the tree H is $\lg N + \lceil \lg(D/N) \rceil = \lceil \lg D \rceil$, and $depth(\kappa)$ is the depth of the root of the maximal-height subtree in which the new corruption site occurs. For $\kappa \geq D/2$, $depth(k) = H$. The validation of the hash chain corresponding to this root evaluates to true before the κth corruption occurs, and false afterwards.

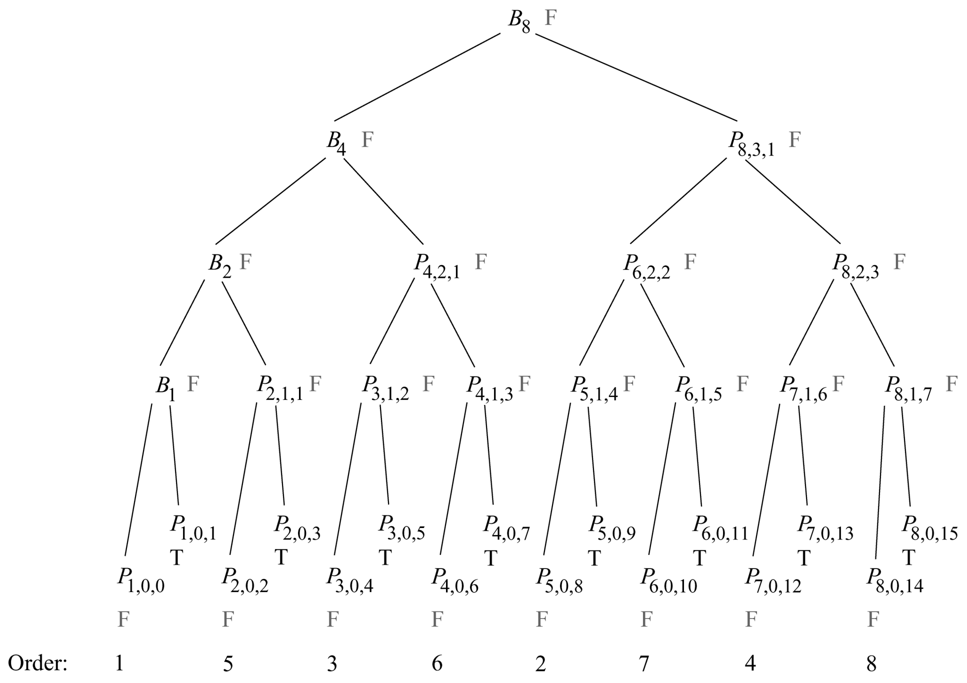


Fig. 25. Worst-case scenario for corruption site distribution (a3D).

The base case for this recursion is $\mathcal{V}(0) = 1$ and corresponds to the case when the result of the validation of the root of the entire tree (B_8 in this case) was true, implying that no corruption has occurred ($\kappa = 0$). For $\mathcal{V}(1)$, there exists a single corruption site in the subtree of maximal height which, in this case, is the entire complete binary tree. This first corruption site corresponds to the number '1' in Figure 25. The corruption site will thus force the validation of the following sequence of hash chains: B_8, B_4, B_2, B_1, $P_{1,0,0}$, $P_{1,0,1}$, $P_{2,1,1}$, $P_{4,2,1}$, $P_{8,3,1}$. The number of chains validated (for a specific κ) is by definition $\mathcal{V}(\kappa)$, hence, $\mathcal{V}(1) = 9$. Alternatively, $\mathcal{V}(1) = \mathcal{V}(0) + 2 \cdot (4 - \lg 1) = 1 + 2 \cdot 4 = 9$. We now solve this recursion.

THEOREM D.1. *The solution to the recursion* $\mathcal{V}(\kappa) = \mathcal{V}(\kappa-1) + 2 \cdot (H - depth)$ *is* $\mathcal{V}(\kappa) = 2 \cdot \kappa \cdot (H - \lceil \lg \kappa \rceil) + (1 + [\kappa \neq 2^i]) \cdot 2^{\lfloor \lg \kappa \rfloor + 1} - 1$, *for some* $i \in \mathbb{N} \cup \{0\}$.

PROOF. The variable *depth* denotes the depth of the root of the maximal-height subtree in which the new corruption occurs. This depth is a function of κ, namely, $depth = \lceil \lg \kappa \rceil$:

$$\begin{aligned}
\mathcal{V}(\kappa) &= \mathcal{V}(\kappa-1) + 2 \cdot (H - depth)\\
&= \mathcal{V}(\kappa-1) + 2 \cdot (H - \lceil \lg \kappa \rceil)\\
&= \mathcal{V}(\kappa-2) + 2 \cdot (H - \lceil \lg(\kappa-1) \rceil) + 2 \cdot (H - \lceil \lg \kappa \rceil)\\
&\vdots\\
&= \mathcal{V}(\kappa-i) + 2 \cdot (H - \lceil \lg(\kappa-(i-1)) \rceil) + \ldots + 2 \cdot (H - \lceil \lg \kappa \rceil).
\end{aligned}$$

So, in order to get a closed form, we unfold the recursion until $\mathcal{V}(\kappa - i) = \mathcal{V}(0)$, which implies $\kappa = i$. We substitute $i = \kappa$ in our recursive formula and get

$$\begin{aligned}
\mathcal{V}(\kappa) &= \mathcal{V}(0) + 2 \cdot (H - \lceil \lg 1 \rceil) + \ldots + 2 \cdot (H - \lceil \lg \kappa \rceil) \\
&= \mathcal{V}(0) + 2 \cdot \sum_{j=1}^{\kappa} (H - \lceil \lg j \rceil) \\
&= \mathcal{V}(0) + 2 \cdot \sum_{j=1}^{\kappa} H - 2 \cdot \sum_{j=1}^{\kappa} \lceil \lg j \rceil \\
&= 1 + 2 \cdot \kappa \cdot H - 2 \cdot \sum_{j=1}^{\kappa} \lceil \lg j \rceil .
\end{aligned}$$

If we expand the last term, we find that $\sum_{j=1}^{\kappa} \lceil \lg j \rceil = 0 + 1 + 2 + 2 + 3 + 3 + 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + \ldots + \lceil \lg \kappa \rceil$. We observe that the terms of the sum can be divided into "groups," each group having the same number repeated a power of 2 number of times. The numbers repeated are 1, 2, 3, up to $\lceil \lg j \rceil$. Thus we can evaluate the original sum by summing the sums of each group, for example, $1 \cdot 1 + 2 \cdot 2^2 + 3 \cdot 2^2 + 4 \cdot 2^3 + \ldots + j \cdot 2^{j-1}$. If κ is a power of 2, then the last "group" of numbers will be complete, otherwise we will a partial final "group." For this reason, we first add up all the complete "groups" in the sum, i.e., $\sum_{j=1}^{\lfloor \lg \kappa \rfloor} j \cdot 2^{j-1}$. Then we add what is left over, i.e., $\kappa - 2^{\lfloor \lg \kappa \rfloor}$ times the last summand which is $\lceil \lg \kappa \rceil$. If we put all the terms together we get:

$$\sum_{j=1}^{\kappa} \lceil \lg j \rceil = \sum_{j=1}^{\lfloor \lg \kappa \rfloor} j \cdot 2^{j-1} + \left(\kappa - 2^{\lfloor \lg \kappa \rfloor}\right) \cdot \lceil \lg \kappa \rceil .$$

We then substitute this sum evaluation into $\mathcal{V}(\kappa)$ and get the following:

$$\begin{aligned}
\mathcal{V}(\kappa) &= 1 + 2 \cdot \kappa \cdot H - 2 \cdot \left(\sum_{j=1}^{\lfloor \lg \kappa \rfloor} j \cdot 2^{j-1} + (\kappa - 2^{\lfloor \lg \kappa \rfloor}) \cdot \lceil \lg \kappa \rceil \right) \\
&= 1 + 2 \cdot \kappa \cdot H - \sum_{j=0}^{\lfloor \lg \kappa \rfloor} j \cdot 2^{j} - 2 \cdot \kappa \cdot \lceil \lg \kappa \rceil + 2^{\lfloor \lg \kappa \rfloor + 1} \cdot \lceil \lg \kappa \rceil .
\end{aligned}$$

The sum $\sum_{j=0}^{\lfloor \lg \kappa \rfloor} j \cdot 2^{j}$ is evaluated using a known formula,

$$\sum_{k=0}^{n} k \cdot x^k = \frac{x - (n+1) \cdot x^{n+1} + n \cdot x^{n+2}}{(1-x)^2}, \qquad \text{for } x \neq 1 .$$

$$\begin{aligned}
\mathcal{V}(\kappa) &= 1 + 2 \cdot \kappa \cdot H - \frac{2 - (\lfloor \lg \kappa \rfloor + 1) \cdot 2^{\lfloor \lg \kappa \rfloor + 1} + \lfloor \lg \kappa \rfloor \cdot 2^{\lfloor \lg \kappa \rfloor + 2}}{(1-2)^2} \\
&\quad - 2 \cdot \kappa \cdot \lceil \lg \kappa \rceil + 2^{\lfloor \lg \kappa \rfloor + 1} \cdot \lceil \lg \kappa \rceil \\
&= 1 + 2 \cdot \kappa \cdot H - 2 + \lfloor \lg \kappa \rfloor \cdot 2^{\lfloor \lg \kappa \rfloor + 1} + 2^{\lfloor \lg \kappa \rfloor + 1} - \lfloor \lg \kappa \rfloor \cdot 2^{\lfloor \lg \kappa \rfloor + 2} \\
&\quad - 2 \cdot \kappa \cdot \lceil \lg \kappa \rceil + 2^{\lfloor \lg \kappa \rfloor + 1} \cdot \lceil \lg \kappa \rceil \\
&= 2 \cdot \kappa \cdot H - 1 + 2^{\lfloor \lg \kappa \rfloor + 1} \cdot (\lfloor \lg \kappa \rfloor + 1 - 2 \cdot \lfloor \lg \kappa \rfloor + \lceil \lg \kappa \rceil) - 2 \cdot \kappa \cdot \lceil \lg \kappa \rceil
\end{aligned}$$

Table XIII. Forensic Areas for $1 \leq \kappa \leq D$ Corruption Sites (a3D)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	N	0	$TotalArea - N$
2	N	0	$TotalArea - N$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
κ	N	0	$TotalArea - N$

$$\begin{aligned}
&= 2 \cdot \kappa \cdot H - 1 + 2^{\lfloor \lg \kappa \rfloor + 1} \cdot (\lceil \lg \kappa \rceil - \lfloor \lg \kappa \rfloor + 1) - 2 \cdot \kappa \cdot \lceil \lg \kappa \rceil \\
&= 2 \cdot \kappa \cdot (H - \lceil \lg \kappa \rceil) + (\lceil \lg \kappa \rceil - \lfloor \lg \kappa \rfloor + 1) \cdot 2^{\lfloor \lg \kappa \rfloor + 1} - 1 \\
&= 2 \cdot \kappa \cdot (H - \lceil \lg \kappa \rceil) + (\,[\kappa \neq 2^i] + 1) \cdot 2^{\lfloor \lg \kappa \rfloor + 1} - 1, \quad \text{for some } i \in \mathbb{N} \cup \{0\}.
\end{aligned}$$

Here we use Iverson brackets [Graham et al. 2004, p. 24]. □

Note that for values of κ between $D/2$ and D, the value of $\mathcal{V}(\kappa)$ is unchanged at $\mathcal{V}(D/2) = 2 \cdot D - 1$. This is because, as we have seen, when κ is equal or exceeds $D/2$, all the hash chains covering the first D days will have to be validated.

Thus we can calculate the cost during forensic analysis quite simply

$$ForensicAnalysis_{a3D} = \mathcal{V}(\kappa)$$

We now examine the breakdown of the three types of areas in the a3D Algorithm. Each granule corresponds to a distinct region of area of height $V \cdot N = N$ (normalized) and width 1 (!) and thus, total $Area_P = \kappa \cdot V \cdot N = \kappa \cdot N$. Moreover, since this algorithm will detect all κ corruption sites, this implies that $Area_U = 0$. The breakdown of the different areas is given in Table XIII.

$$\begin{aligned}
FC_{a3D}(D, N, 1, \kappa) &= (D/N + \mathcal{N}(D) + D/N - (1 + \lfloor \lg(D/N) \rfloor) + \mathcal{V}(\kappa)) \\
&\quad + Area_P \\
&= (D/N + 2 \cdot D - 1 + D/N - (1 + \lfloor \lg(D/N) \rfloor) \\
&\quad + 2 \cdot \kappa \cdot (\lceil \lg D \rceil - \lceil \lg \kappa \rceil) + (1 + [\kappa \neq 2^i]) \cdot 2^{\lfloor \lg \kappa \rfloor + 1} - 1) \\
&\quad + (\kappa \cdot N) \qquad (6) \\
&= O(\kappa \cdot N + D + \kappa \cdot \lg D)
\end{aligned}$$

In the case of the a3D Algorithm, we see an increase in the cost of normal processing and forensic analysis, but the area produced by this algorithm is optimal. $Area_U$ is zero, while $Area_P$ achieves its minimum because, by definition, each granule cannot be shrunk below the spatial resolution R_s (as discussed in Section 8.4). For a reason why the *temporal* dimension of the area, that is, the uncertainty of t_c, cannot be shrunk further, see Section 10.

Finally, when $\kappa > D/2$, rather than doing κ binary searches, we can simply scan the R_r units, reducing the forensic cost to $O(\kappa \cdot N + D + D) = O(\kappa \cdot N)$.

E. FORENSIC COST FOR BEST-CASE DISTRIBUTION OF CORRUPTION SITES

We perform an analysis of the forensic cost of the four algorithms assuming a *best-case* distribution of κ corruption sites.

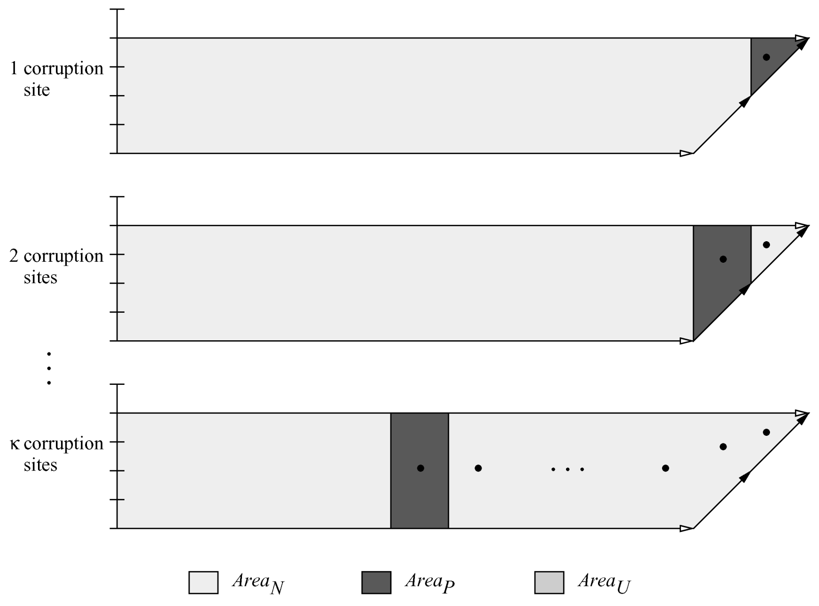


Fig. 26. Three types of forensic area for best-case distribution of κ corruption sites (Monochromatic).

Table XIV. Forensic Areas for Best-case Distribution of κ Corruption Sites (Monochromatic)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	V	0	$TotalArea - V$
2	V	0	$TotalArea - V$
3	V	0	$TotalArea - V$
⋮	⋮	⋮	⋮
κ	V	0	$TotalArea - V$

E.1 Monochromatic Algorithm

A best-case distribution for the Monochromatic algorithm occurs when each of the κ corruption sites appears in a different notarization interval at the rightmost end of the trapezoid in the corruption diagram. This means that the sites occur starting from the most recent I_N and going back to older notarization intervals in a contiguous manner, as shown in Figure 26. As in Section 9, we examine how each corruption site partitions the trapezoid—bound by the last validation event—into the three types of forensic area, that is, $Area_P$, $Area_U$, and $Area_N$. Observe that, unlike in the worst-case distribution, the corruption sites are examined from right to left. This, in conjunction with the fact that only one corruption site occurs within each notarization interval, allows each site to be positively identified. Hence Table XIV shows that each site is associated with an $Area_P$ but not with an $Area_U$.

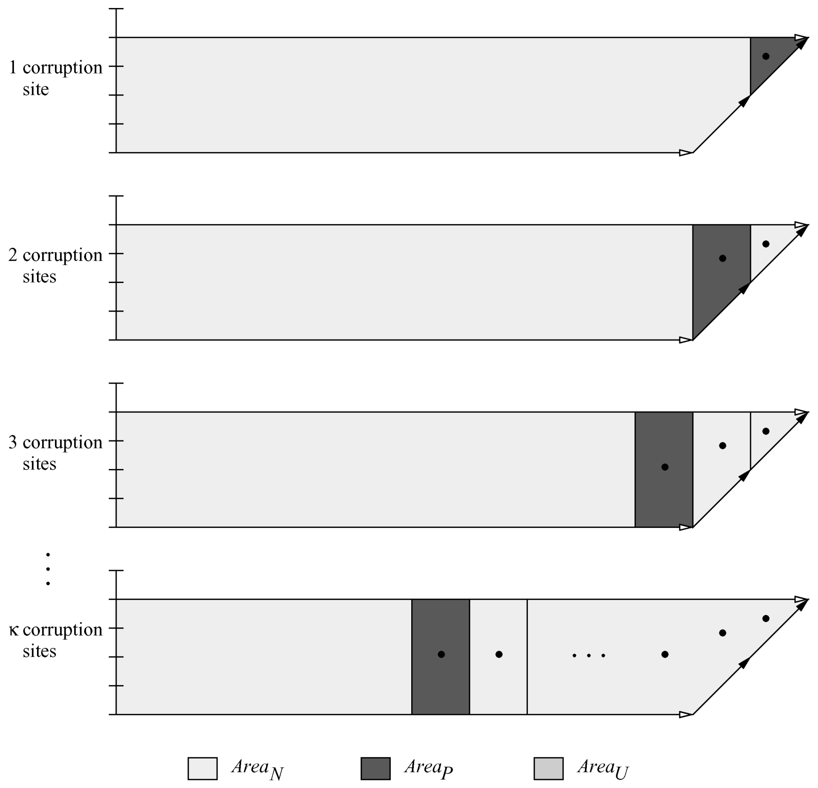


Fig. 27. Three types of forensic area for best-case distribution of κ corruption sites (RGBY).

All the terms in the forensic cost formula remain the same, as in the worst-case, except for the forensic areas. Summing $Area_P$ over all corruption sites, we can compute the forensic cost for the Monochromatic Algorithm:

$$\begin{aligned} FC_{mono}(D, 1, V, \kappa) &= (D + D/V + 2 \cdot \lg D) \\ &\quad + \left(V + \sum_{i=2}^{\kappa} V \right) \\ &= D + D/V + 2 \cdot \lg D + V + (\kappa - 1) \cdot V \\ &= O(\kappa \cdot V + D). \end{aligned}$$

The forensic cost of the Monochromatic Algorithm for best-case distribution is asymptotically smaller than cost for worst-case distribution, which is $O(\kappa \cdot V \cdot D)$.

E.2 RGBY Algorithm

In the case of the RGBY Algorithm, the *best*-case distribution of corruption sites is exactly the same as in the Monochromatic Algorithm. The sites occur starting from the most recent I_N and going back to older notarization intervals in a contiguous manner, as shown in Figure 27. Once again, because of the

Table XV. Forensic Areas for Best-case Distribution of κ Corruption Sites (RGBY)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	2	0	$TotalArea - 2$
2	2	0	$TotalArea - 2$
3	2	0	$TotalArea - 2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
κ	2	0	$TotalArea - 2$

assumption of only one site per I_N, we have only positive areas associated with each site.

Table XV shows the breakdown of the three types of the forensic areas for each of the κ corruption sites. All the terms in the forensic cost formula remain the same as in the worst case, except for the forensic areas. Summing $Area_P$ over all corruption sites, we can compute the forensic cost for the RGBY Algorithm:

$$\begin{aligned} FC_{RGBY}(D, 1, 2, \kappa) &= (D + 3 \cdot (D/2) + 2 \cdot (D/2) + 2 \cdot \lg D) \\ &\quad + \left(\sum_{i=1}^{\kappa} 2 \right) \\ &= O(\kappa + D). \end{aligned}$$

The forensic cost of the RGBY Algorithm for best-case distribution is asymptotically the same as the cost for worst-case distribution, which is also $O(\kappa + D)$.

E.3 Tiled Bitmap Algorithm

The best-case distribution for the Tiled Bitmap Algorithm happens when the corruption sites occur one in each tile, tampering the last granule in the tile, as shown in Figure 28. The resulting bit string uniquely identifies the corrupted granule, so we can positively identify the corruption site in the tile with no false positives.

Table XVI shows that each $Area_P$ has area $V \cdot N$, and thus summing over all corruption sites yields the new forensic cost.

$$\begin{aligned} FC_{tiled\_bitmap}(D, N, V, \kappa) &= ((D/N) + (1 + \lg N) \cdot (D/N) + D/(V \cdot N) \\ &\quad 2 \cdot \lg(D/N) + (1 + \lg N) \cdot \kappa) + (\kappa \cdot V \cdot N) \\ &= O(\kappa \cdot V \cdot N + (D \cdot \lg N)/N + \lg D) \end{aligned}$$

The new cost is asymptotically lower than the corresponding cost for worst-case distribution. In particular, the $\kappa \cdot V \cdot N^2$ term in the worst-case cost has lost a factor of N.

E.4 a3D Algorithm

The best-case distribution of corruption sites for the a3D Algorithm, is one in which the sites occur consecutively in leaves of the tree as shown in Figure 29. The numbers labeled with "Order of κ" show the order with which the sites are

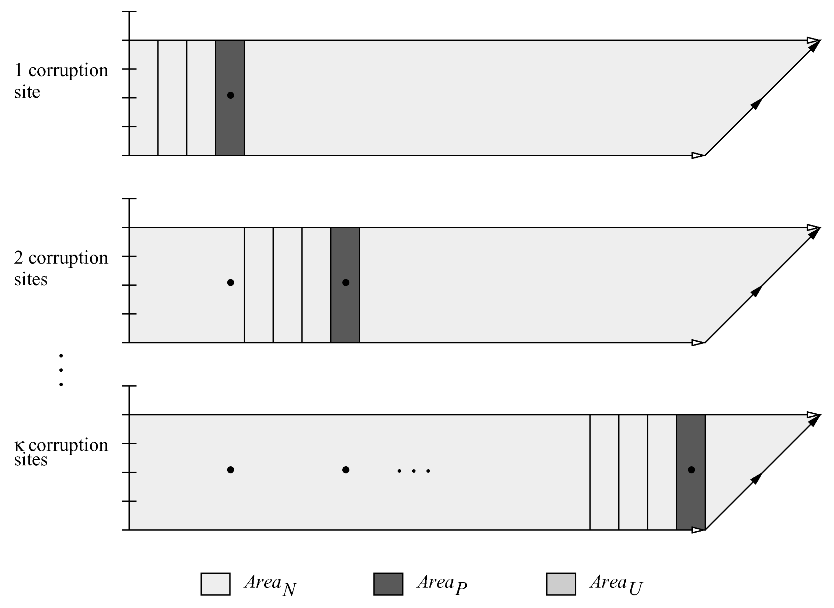


Fig. 28. Three types of forensic area for best-case distribution of κ corruption sites (Tiled Bitmap).

examined. This order is such that moving from one site to the next incurs a minimum increase in the number of chains validated in the tree. This defers the validation of the entire tree until the last (16th) site has been examined.

The number of validations in the forensic analysis with each successive corruption site satisfies the following recursive formula,

$$\mathcal{V}_b(\kappa) = \mathcal{V}_b(\kappa - 1) + 2 \cdot (H - depth_b(\kappa)) \qquad \text{for } 1 \leq \kappa \leq D\ , \tag{7}$$

where $\mathcal{V}_b(\kappa)$ is the number of hash chains validated by the algorithm when κ corruption sites exist under a *best*-case distribution, the height of the tree H is $\lg N + \lceil \lg(D/N) \rceil = \lceil \lg D \rceil$, and $depth_b(\kappa)$ is the depth of the root of the maximal-height subtree in which the new corruption site occurs, again, under a best-case distribution. The validation of the hash chain corresponding to this root evaluates to true before the κ^{th} corruption occurs, and false afterwards.

To find how $depth_b$ depends on κ, we first number the nodes of the tree in a breadth-first manner. The numbers are shown in `courier` font in Figure 29. Observe that the leaves under this numbering scheme are labeled by p and the correspondence between p and κ is $p = \kappa + D - 1$. Observe also that we only need to deal with leftmost paths of subtrees, since any site occurring in a leaf of a rightmost path contributes zero to $\mathcal{V}_b$. Recall that, for all nodes in a binary tree, if the parent has index i, its children have indices $2 \cdot i$ and $2 \cdot i + 1$. Hence, in order to find the number of the root (inner node) of the subtree given a p number of a leftmost leaf, we must divide p by 2^x, where x is the maximum integer such that $2^x \mid p$ but $2^{x+1} \nmid p$. In other words, x is the zero-based index, counting from the right end of the leftmost "1" in the binary representation of p. We can use Iverson brackets [Graham et al. 2004, p. 24] to express x as the sum $x = \sum_{1 \leq l \leq \lfloor \lg p \rfloor} [p \bmod 2^l = 0]$. Given the position of the root, we can find

Table XVI. Forensic Areas for Best-case Distribution of κ Corruption Sites (Tiled Bitmap)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	$V \cdot N$	0	$TotalArea - V \cdot N$
2	$V \cdot N$	0	$TotalArea - V \cdot N$
3	$V \cdot N$	0	$TotalArea - V \cdot N$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
κ	$V \cdot N$	0	$TotalArea - V \cdot N$

its depth by the following formula:

$$depth_b(\kappa) = \left\lfloor \lg\left(\frac{p}{2^x}\right) \right\rfloor, \tag{8}$$

where $p = \kappa + D - 1$ and $x = \sum_{1 \leq l \leq \lfloor \lg p \rfloor} [p \bmod 2^l = 0]$.

We can now substitute (8) in the recursion, unfold it, and get a closed form:

$$\mathcal{V}_b(\kappa) = 1 + 2 \cdot \kappa \cdot \lg D - 2 \cdot \sum_{i=1}^{\kappa} \left\lfloor \lg\left((p-i)/\left(2^{\sum_{1 \leq l \leq \lfloor \lg(p-i) \rfloor} [(p-i) \bmod 2^l = 0]}\right)\right) \right\rfloor.$$

We do not attempt to evaluate the sum, but we rather try to find the asymptotic upper bound for $\mathcal{V}_b$. The idea is to minimize the value of the sum so that the entire expression can be bounded from above. The minimum value the numerator in the summand can take is $D - 1$ when $i = \kappa$, while the maximum value the denominator can take is $2^H = D$, that is, when we are considering the root of the entire tree. This makes the sum easy to bound as shown in equation (9):

$$\begin{aligned}
\mathcal{V}_b(\kappa) &= 1 + 2 \cdot \kappa \cdot \lg D - 2 \cdot \sum_{i=1}^{\kappa} \lfloor \lg\left((p-i)/(2^{\sum_{1 \leq l \leq \lfloor \lg(p-i) \rfloor} [(p-i) \bmod 2^l = 0]})\right) \rfloor \\
&\leq 1 + 2 \cdot \kappa \cdot \lg D - 2 \cdot \sum_{i=1}^{\kappa} \lg((D-1)/D) \\
&\leq 1 + 2 \cdot \kappa \cdot \lg D - 2 \cdot \kappa \lg((D-1)/D) \\
&\leq 1 + 2 \cdot \kappa \cdot \lg D + 2 \cdot \kappa \lg(D/(D-1)) \\
&\leq 1 + 2 \cdot \kappa \cdot \lg D + 2 \cdot \kappa \cdot \lg D \Rightarrow \\
\mathcal{V}_b(\kappa) &= O(\kappa \cdot \lg D).
\end{aligned} \tag{9}$$

Putting everything together, we can now evaluate the best-case forensic cost of the a3D Algorithm:

$$\begin{aligned}
FC_{a3D}(D, N, 1, \kappa) &= (D/N + \mathcal{N}(D) + D/N - (1 + \lfloor \lg(D/N) \rfloor) + \mathcal{V}(\kappa)) \\
&\quad + Area_P \\
&= (D/N + 2 \cdot D - 1 + D/N - (1 + \lfloor \lg(D/N) \rfloor) \\
&\qquad + \kappa \cdot \lg D) \\
&\qquad + (\kappa \cdot N) \\
&= O(\kappa \cdot N + D + \kappa \cdot \lg D).
\end{aligned}$$

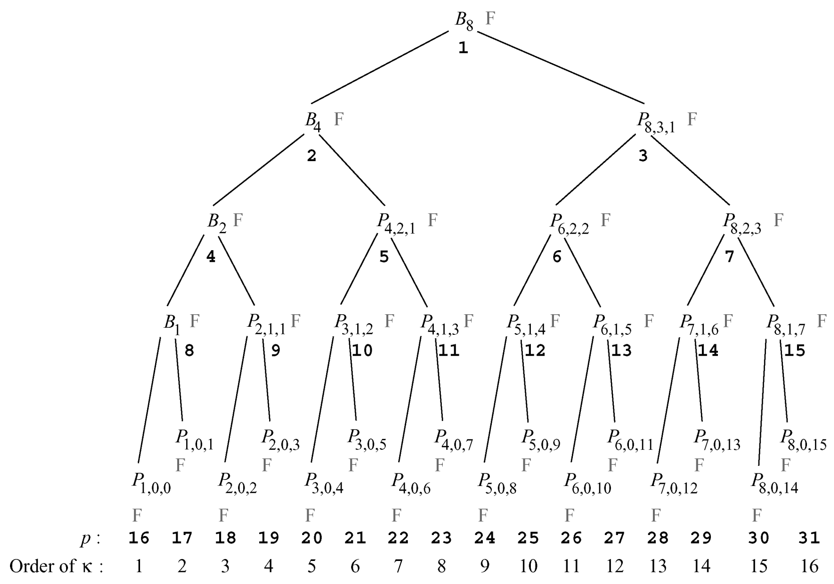


Fig. 29. Three types of forensic area for best-case distribution of κ corruption sites (a3D).

Table XVII. Forensic Areas for Average-Case Distribution of κ Corruption Sites (Monochromatic)

# Corruption Sites ($1 \leq \kappa \leq D$)	$Area_P$	$Area_U$	$Area_N$
1	V	0	$TotalArea - V$
2	0	$(2 \cdot TotalArea - V)/3$	$(TotalArea + V)/3$
3	0	$(3 \cdot TotalArea - V)/4$	$(TotalArea + V)/4$
⋮	⋮	⋮	⋮
κ	0	$(\kappa \cdot TotalArea - V) \cdot \frac{1}{\kappa+1}$	$(TotalArea + V) \cdot \frac{1}{\kappa+1}$

The asymptotic forensic cost for the worst-case distribution is thus identical to that for the best-case distribution of a large number of corruption sites, namely, $O(\kappa \cdot N + D + \kappa \cdot \lg D)$.

F. FORENSIC COST FOR AVERAGE-CASE DISTRIBUTION OF CORRUPTION SITES

In this section, we give an analysis of the forensic cost of the four algorithms, assuming an average distribution of κ corruption sites. The analysis for the Monochromatic and RGBY Algorithms are similar in approach and detail, that, for each corruption site, we examine how it partitions the trapezoid bound below by the last validation event, into the three types of forensic area $Area_P$, $Area_U$, and $Area_N$. However, to obtain an estimate of the forensic cost of the Tiled Bitmap Algorithm, we employ the average size of the candidate set instead of considering the distribution of the corruption sites. In the case of the a3D Algorithm, the analysis is much simpler, since we have shown that the forensic cost is the same for best and worst case distributions of corruption sites.

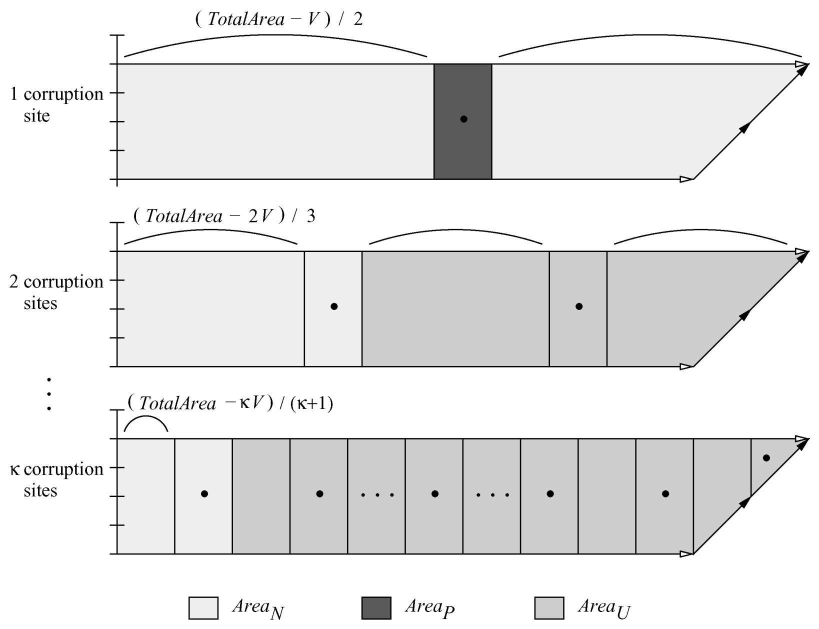


Fig. 30. Three types of forensic area for average-case distribution of κ corruption sites (Monochromatic).

F.1 Monochromatic Algorithm

In order to obtain a bound on the forensic cost of the Monochromatic Algorithm, we assume that the κ corruption sites are evenly distributed in the trapezoid, as shown in Figure 30. Each successive addition of a corruption site splits the area evenly and hence, if there are κ sites, then each intervening area between them has size $(\mathit{TotalArea} - \kappa \cdot V)/(\kappa + 1)$.

Figure 30 shows that only the first corruption site can be positively identified as was true in the worst-case distribution. We consider the forensic cost of a single corruption site ($\kappa = 1$) separately from the cases where $\kappa > 1$, the reason being that the area breakdown is different in the two cases. Notice that for $\kappa \geq 2$, the last term in the cost formula (10) is a partial sum of a harmonic series. It is an established result [Graham et al. 2004, p. 276] that a partial sum of the harmonic series H_n is bounded above by $\lfloor \lg n \rfloor + 1$.

$$\begin{aligned}
FC_{mono}(D, 1, V, 1) &= (D + D/V + 2 \cdot \lg D) + V \\
&= O(V + D) \\
FC_{mono}(D, 1, V, \kappa \geq 2) &= (D + D/V + 2 \cdot \lg D) \\
&\quad + \left(V + \sum_{i=2}^{\kappa} (i \cdot \mathit{TotalArea} - V)/(i+1) \right) \\
&\leq D + D/V + 2 \cdot \lg D + V + (\kappa - 1) \cdot \mathit{TotalArea} \\
&\quad - V \cdot \sum_{i=2}^{\kappa} 1/(i+1) \qquad (10)
\end{aligned}$$

$$\begin{aligned}&\le D + D/V + 2\cdot \lg D + V + (\kappa - 1)\cdot \mathit{TotalArea}\\&\quad - V\cdot(\lfloor \lg(\kappa+1)\rfloor - 1/2)\\&= O(\kappa\cdot V\cdot D)\end{aligned}$$

The forensic cost of the Monochromatic Algorithm for the average case distribution is asymptotically the same as the cost for worst-case distribution.

F.2 RGBY Algorithm

The forensic cost of the RGBY Algorithm for the worst-case distribution of κ corruption sites is asymptotically the same as the one for best-case distribution: $O(\kappa + D)$. This implies that the forensic cost for the average case distribution of corruption site is the same.

F.3 Tiled Bitmap Algorithm

To obtain an estimate of the forensic cost of the Tiled Bitmap Algorithm, we do not consider the distribution of the κ corruption sites. Rather, for each site we must deduce its relative position within a tile so that the size of the candidate set can be computed. Furthermore, given a uniform distribution of κ, we have no way of enforcing that each site will belong to a different tile. For these reasons, we consider the average size of the candidate set instead.

LEMMA 1. *The average cardinality of the candidate sets for $k = 2$ and for a given $l = \lg N$ is $\overline{|C|} = \frac{3^l - 1}{2^l}$.*

PROOF. The average is $\overline{|C|} = \frac{1}{2^l}\cdot((\sum_{z=0}^{l}\binom{l}{z}\cdot 2^z) - 1)$. $\sum_{z=0}^{l}\binom{l}{z}\cdot 2^z$ is the binomial expansion of $(2+1)^l = 3^l$. So $\overline{|C|} = \frac{3^l-1}{2^l}$. □

Note that $\overline{|C|} = \frac{3^l-1}{2^l} < 1.5^l = O(1.5^l)$. For $l = 10$, a candidate set will contain on average about 5% of the possible binary numbers of length l. For $l > 20$, a candidate set will contain on average only about 0.3% of the possible strings. This is expected, since the fraction $\frac{1.5^l}{2^l}$ decreases as l increases.

This decrease in candidate set cardinality as l increases has implications for forensic analysis. Recall that the goal is to determine the set of possible corruption events implied by a provided target binary number. While the number of possibilities grows as l gets larger, the percentage of possible granules declines.

We have showed that the average cardinality of all possible candidate sets for a fixed-length *target* is $\overline{|C|} = (3^l - 1)/2^l$. Recall that $l = \lg N$.

$$\begin{aligned}Area_P &= \sum_{i=1}^{\kappa}\overline{|C|}\cdot V\cdot N = \kappa\cdot\frac{3^{\lg(N)}-1}{2^{\lg(N)}}\cdot V\cdot N = \kappa\cdot\frac{3^{\lg_3(N)/\lg_3 2}-1}{N}\cdot V\cdot N\\&= \kappa\cdot V\cdot(N^{\lg 3} - 1)\end{aligned}$$

Thus the forensic cost of the algorithm, taking the average cardinality of the candidate set, is:

$$\begin{aligned}FC_{tiled\_bitmap}(D,N,V,\kappa) &= ((D/N) + (1+\lg N)\cdot(D/N) + D/(V\cdot N)\\&\quad + 2\cdot\lg(D/N) + (1+\lg N)\cdot\kappa)\end{aligned}$$

$$
\begin{aligned}
&+ (\kappa \cdot V \cdot (N^{\lg 3} - 1)) \\
&= O(\kappa \cdot V \cdot N^{\lg 3} + (D \cdot \lg N)/N + \lg D)\,.
\end{aligned}
$$

This replaces a factor of N^2 with $N^{\lg 3}$, making the average cost asymptotically lower than in the worst case.

F.4 a3D Algorithm

The forensic cost of the a3D Algorithm for the worst-case distribution of κ corruption sites is asymptotically the same as the one for best-case distribution: $O(\kappa \cdot N + D + \kappa \cdot \lg D)$. This implies that the forensic cost for the average case distribution of corruption site is the same.