A Group Testing Approach to Improved Corruption Localizing Hashing

Annalisa De Bonis[1] and Giovanni Di Crescenzo[2]

[1] Università di Salerno, Fisciano, Salerno, Italy. E-mail: debonis@dia.unisa.it
[2] Telcordia Technologies, Piscataway, NJ, USA. E-mail: giovanni@research.telcordia.com

Abstract. Efficient detection of integrity violations is crucial for the reliability of both data at rest and data in transit. While ideally one would want to always find all changes in the input data, in practice this capability may be expensive, and one may be content with localizing or finding a superset of any changes. Corruption-localizing hashing [5] is a cryptographic primitive that enhances collision-intractable hash functions thus improving the detection property of these functions into a suitable localization property. Corruption localizing hash schemes obtain some superset of the changes location, where the accuracy of this superset with respect to the actual changes is a metric of special interest, called localization factor. In this paper we consider the problem of designing corruption localizing hash schemes with reduced localization factor. In [2], combinatorial group testing techniques have been exploited to construct the first corruption-localizing hash scheme with *constant* localization factor and sublinear storage and time complexity. In this paper we continue the approach of [2] and present schemes that improve on previous constructions in that they achieve an arbitrarily small localization factor while insuring the same time and storage complexity of the schemes in [2].

Keywords: Algorithms, Cryptography, Corruption Localizing Hashing, Group Testing, Superimposed Codes.

1 Introduction

Efficient detection of integrity violations is crucial for the reliability of both data at rest and data in transit. Data integrity may be compromised by several factors including unintended memory or hardware faults, or malicious intrusions by an attacker interested in performing unauthorized modifications to the data. Practical scenarios where maintaining data integrity is of special interest include: sensitive data stored on various types of computer memories or backed up in storage facilities, and data or software downloaded from the Internet web sites. In these scenarios, it would be very desirable to have the capability of obtaining information about the location of corrupted data blocks. For instance, in the case of file download from a web site, localizing the corruptions would avoid to repeat the entire download procedure since only part of the data would need to be retransmitted. Similarly, in the case of corruption of stored information, the cost of data recovery could be highly reduced if the virus diagnostic procedure, typically launched after the data is detected to be corrupted, could concentrate on a small area of infected data. Just as collision-intractable hash functions allow to detect undesired changes in the stored or transmitted data, corruption-localizing hash schemes target the localization (i.e., obtaining information about the actual location) of the corruption, via hashing techniques.

1.1 Previous work and related areas

In [5, 6] it has been shown that collision-intractable hash functions can be used not only to detect unexpected modifications in data but also to localize the corrupted area in the modified data. To this aim, the authors of [6] have introduced virus-localizing schemes based on cryptographic hashing. In their model, virus-localizing schemes work under the hypothesis that all corrupted blocks occur within a single segment of the given data. The ideas put forward in [6] have been further developed by the authors of [5] who have formally defined corruption-localizing hash schemes as an extension of collision intractable hash schemes, and generalized the concept of virus-localization to the case of an arbitrary number of corrupted segments. In [12] a similar problem of localization via randomized techniques was studied in conjunction with popular data structures and associated space restrictions. In all three mentioned papers, there is an extensive discussion of the differences of these problems with related but distinct research areas, including coding theory, digital watermarking, software download, program checking, memory correctness checking, combinatorial group testing and authenticated data structures. These papers also discuss the conceptual relationships between the notion of localization, with respect to the notions of detection and correction.

The paper [2] continued the research approach of [5, 6, 12]. In particular, that paper addresses the problem of localizing multiple corruptions by only paying a very small performance price, and obtain a scheme with a constant localization factor and storage and time complexity proportional to $O(v(\log v)\log n + v(\log\beta)\log(n/v))$, where n is the length of the input message, v is an *upper bound* on the number of corruptions, and β is an *upper bound* on length of the longest corrupted segment. The closest previous results include: [6], presenting a scheme based on conventional (non-keyed) hash functions that localizes a single corrupted segment with constant localization factor and storage and time complexity proportional to $O(\log n)$, and [5], presenting a scheme based on conventional hash functions that, when v is constant (as a function of n), achieves $O(n^d)$ localization factor and storage and time complexity proportional to $O(\log^2 n)$, for any $0 < d < 1$. The paper [5] also presents a scheme based on keyed hash functions that achieves $O(v^3)$ localization factor and storage and time complexity proportional to $O(v^2(\log^{1+\epsilon} n)\log_v n)$, for any $\epsilon > 0$.

1.2 Summary of Results

Our schemes are based on a notion of localizing codes recently introduced in [2]. Localizing codes are used in conjunction with collision-intractable hashing to obtain corruption-localizing hashing. We present constructions of localizing codes based on efficient solutions for a variant of group testing, targeting the localization (rather than exact determination) of multiple arbitrary-length (rather than length-1) corruptions.

The analysis of corruption-localizing hash schemes is based on two conflicting metrics: the localization factor that measures the accuracy of localization, and the tag length that captures the efficiency of the scheme in terms of storage and time complexity. Our goal is to design localizing codes that achieve the smallest possible localization factor and tag length, while keeping an arbitrary $\beta = O(n)$ and an unbounded v.

In Section 5 we present a construction that results in a localizing-corruption hash scheme that can be instantiated to achieve a localization factor *arbitrarily close* to β and storage and time complexity $O(v^2\log(n/(\beta v)))$. This localizing code is used in Section 6 as a building block to design an improved localizing code with localization factor *arbitrarily close* to 1, and storage and time complexity proportional to $O(v(\log v)\log n + v(\log\beta)\log(n/v))$. We remark that we assume that the exact number of corruptions as well as the maximum length of corruptions is not given, in that v and β are only upper bounds on the number and length of corruptions.

Our scheme is the first scheme with sublinear storage and time complexity that achieves a localization factor arbitrarily close to 1. Previous efficient schemes with constant localization include the scheme in [2] that achieves a constant localization factor larger than 6. Other than this, previously known schemes achieve constant localization factor only under the following model restrictions: in the case $\beta = 1$ and $v = O(1)$ [4], in the case $\beta = 1$ and v bounded by some polynomial in n [12], in the case $v = 1$ by [6], and when $v = O(1)$ [5].

2 Model and Preliminaries

In this section we review formal notions and definitions of interest in the rest of the paper; specifically: collision-intractable hashing, corruption-localizing hashing, combinatorial group testing and superimposed codes.

2.1 Collision-intractable Hashing.

Informally speaking, a family of collision-intractable hash functions is a family of functions that satisfy the following properties: there exists an efficient algorithm sampling a function from the family; there exists an efficient algorithm computing the output of each function sampled from the family; and no efficient algorithm can find two preimages that are mapped to the same image by any function sampled from the family, unless with small probability. Interestingly, this latter property, also called collision-intractability, holds even when inputs are much longer than (typically, constant-size) outputs. We now proceed more formally.

Let $\mathcal{H} = \{H_\lambda\}_{\lambda\in\mathcal{N}}$, with H_λ being a set of functions $h_\lambda : \{0,1\}^{p(\lambda)} \rightarrow \{0,1\}^\sigma$, where λ is a security parameter, p is a polynomial and σ is constant with respect to λ. A *family of hash functions* $\mathcal{H}$, is a family of polynomial-time (in λ) samplable and computable functions h_λ that take a $p(\lambda)$-bit *message* as input and return a σ-bit *tag*. We denote as $t_n(H)$ the running time of hash functions in H_λ on inputs of length n. We now recall the formal definition of collision intractability.

Definition 1. *Let $\mathcal{H} = \{H_\lambda\}_{\lambda\in\mathcal{N}}$ be a family of hash functions. For any $t, \epsilon > 0$, we say that $\mathcal{H}$ is (t,ϵ)*-collision-intractable *if for any algorithm A running in time t,*

$$\text{Prob}[\, h_\lambda \leftarrow H_\lambda; (x_1, x_2) \leftarrow A(1^\lambda, h_\lambda) \,:\, x_1 \neq x_2 \wedge h_\lambda(x_1) = h_\lambda(x_2)\,] \leq \epsilon,$$

where the notation $\leftarrow$ denotes a random process (e.g., randomly choosing from a fixed set, or from the set of outputs of an algorithm on a given input).

Collision-intractable hashing is used in several real-life applications, to detect whether an input message was modified or not. Constructions in the literature are based on either the computational intractability of number-theoretic problems (see, e.g., [1]), or more general complexity-theoretic assumptions (see, e.g., [14]), or heuristic finite functions with very high efficiency but only conjectured collision intractability.

2.2 Corruption-Localizing Hashing.

Informally speaking, a corruption-localizing hash scheme is a pair of algorithms, a hashing algorithm and a localizer that satisfy the following properties: both algorithms run in deterministic polynomial time; the hashing algorithm can be seen as a collision-intractable hash function; the localizer, when given as input the hash tag for the original input string, the corrupted input and the tolerated number of errors, returns a superset of these errors that is larger than the number of errors by a given multiplicative factor, called the localization factor, unless with small probability. We now proceed more formally, building on the model from [6, 5].

Data model. Let x denote an n-bit message (extending our results to messages consisting of n blocks of multiple bits as atomic components is immediate). For any two block indices $i, j \in \{1 \ldots, n\}$ with $i \leq j$, we denote by $x[i]$ the i-th block of x and by $x[i, j]$ the sequence of consecutive bits $x[i], x[i+1], \ldots, x[j-1], x[j]$ with indices in the interval $[i, j]$. A sequence of consecutive bits of a message x will be referred to as a *segment* of x. A sequence of segments $\{[x_{i_1}, x_{j_1}], \ldots [x_{i_k}, x_{j_k}]\}$, with $i_1 \leq j_1 \leq i_2 \leq j_2, \ldots \leq i_k \leq j_k$, will be referred to as *segment list.*

Adversary model. Given two n-bit messages x and x', we measure their difference using function $\mathsf{Diff}_v[x, x'] = \min \sum_{r=1}^{v} |S_r|$, where each S_r, for $r = 1, \ldots, v$, is an interval in $[0, n-1]$, and the minimum is taken over all $S_1, \ldots, S_v$ such that $x[\overline{\bigcup_{r=1}^{v} S_r}] = x'[\overline{\bigcup_{r=1}^{v} S_r}]$[3]. Notice that S_r might be empty for some $r \in \{1, \ldots, v\}$. If $S_1, \ldots, S_v$ are v intervals in $[0, n-1]$ such that $\mathsf{Diff}_v[x, x'] = \sum_{r=1}^{v} |S_r|$ and $x[\overline{\bigcup_{r=1}^{v} S_r}] = x'[\overline{\bigcup_{r=1}^{v} S_r}]$ then we say that $S_1, \ldots, S_v$ *achieve* $\mathsf{Diff}_v[x, x']$. Intuitively, $\mathsf{Diff}_v[x, x']$ represents the minimum total size of up to v segments that an adversary has to modify in order to change x into x'. We say that an n-bit vector e is a (β, v)*-corruption vector* if there exists $|S_i| \leq \beta$, for $i = 1, \ldots, v$, where $S_1, \ldots, S_v$ achieve $\mathsf{Diff}_v[0^n, e]$. Note that $\mathsf{Diff}_v[x, x'] = \mathsf{Diff}_v[0^n, e]$, where $e = x \oplus x'$, 0^n denotes the n-bit zero vector and $\oplus$ denotes the logical XOR operation. We can then characterize the adversary's attack as issuing two messages x, x' such that $e = x \oplus x'$ is a (β, v)-corruption vector and formally define corruption-localizing hash schemes as follows.

Definition 2. *A* hash scheme *is a pair of algorithms $HS = (clH, Loc)$, with the following syntax. On input an n-bit string x, algorithm clH returns a string tag. On input a positive integer v, an n-bit string x' and a string tag, algorithm Loc returns a set of indices $T \subseteq \{0, \cdots, n-1\}$. Both algorithms are deterministic and run in time polynomial in some security parameter λ. We say that the hash scheme* HS *is* $(t, \epsilon, \alpha, \beta, v)$-corruption-localizing *if for any algorithm A running in time t and returning distinct x, x', it holds that $p(\mathsf{Succ}_1(A; \mathrm{HS}; \alpha, v)) \leq \epsilon$, where probability $p(\mathsf{Succ}_1(A; \mathrm{HS}; \alpha, v))$ is formally defined as* $\Pr\left[(x, x') \leftarrow A(1^\lambda) : clH(x) = clH(x')\right]$*, and if whenever $e = x \oplus x'$ is a (β, v)-corruption vector, then $p(\mathsf{Succ}_2(A; \mathrm{HS}; \alpha, v)) \leq \epsilon$, where probability $p(\mathsf{Succ}_2(A; \mathrm{HS}; \alpha, v))$ is formally defined as*

$$\Pr\left[(x, x') \leftarrow A(1^\lambda); T \leftarrow Loc(v, x', clH(x)) : (x[\overline{T}] \neq x'[\overline{T}]) \vee \left(\frac{|T|}{\mathsf{Diff}_{v}[x, x']} > \alpha\right)\right].$$

[3] We note that our constructions do not require an efficient algorithm for computing $\mathsf{Diff}_v[x, x']$.

We note that the adversary is successful if it either finds a collision to the hashing algorithm clH or prevents *effective localization* (i.e., one of the modified bits is not included in T, and thus $x[\overline{T}] \neq x'[\overline{T}]$), or forces the scheme to exceed the expected localization factor (i.e., $|T| > \alpha \cdot \mathsf{Diff}_v[x, x']$). We use the following metrics to design and analyze corruption-localizing hash schemes: the hashing algorithm and localizer's *running times*; the *localization factor* α, defined as $|T|/\mathsf{Diff}_v[x, x']$; and the *tag length*, defined as the length of the output tag from algorithm clH. In [5] it was also noted that two trivial schemes exist that achieve (1) $\alpha = n/v$, or (2) $\alpha = 1$ and tag length $O(n)$, thus moving the scheme design target to achieving $\alpha = o(n/v)$ and tag length $o(n)$. As in our schemes, the output tag from algorithm clH can be split into message-independent components (i.e., the description of a binary code and the description of a collision-intractable hash function) and message-dependent components (i.e., the hash tags), we define the length of these two quantities as the *off-line tag length* and the *on-line tag length*, respectively. Furthermore, we simplify the on-line tag length metric by using instead the *number of hash tags*, defined as the number of hash tags from a collision-intractable hash function contained in the message-dependent component of tag. In our constructions both the hashing algorithm and localizer's running times are efficient, and the off-line tag length is sublinear in n; thus, the main metrics of interest will be the localization factor and the number of hash tags.

2.3 Group Testing and Superimposed Codes

Combinatorial group testing [7] is the problem of searching the *positive* elements of a given set $\mathcal{O}$ of n elements by posing queries of the form "Does $\mathcal{Q}$ contain any positive element?", with $\mathcal{Q}$ being a subset of $\mathcal{O}$. Search strategies where all queries are decided in advance are called *non-adaptive strategies*, as opposed to *adaptive strategies* where the queried subsets can be chosen after looking at the answers to the previous queries. Typically, non-adaptive strategies are far more costly than adaptive strategies. With respect to combinatorial group testing, it is well known that the best adaptive strategies attain the $\Omega(\log \binom{n}{q})$ *information theoretic lower bound*, with q being an upper bound on the number of positive items. On the other hand, the number of queries needed to determine up to q positive elements by a non-adaptive strategy is lower bounded by $\Omega(\frac{q^2}{\log q} \log(n/q))$. Indeed, it is well known that the cost of non-adaptive strategies for group testing corresponds to the length of $(1, q)$-superimposed codes [10, 13]. In the following, we recall a more general definition of superimposed codes.

A *binary code* of size n and length N is an $N \times n$ matrix $C = \{C(i, j)\}$ with $C(i, j) \in \{0, 1\}$, whose columns are called *codewords*. Given two equal-length binary vector x, y, we say that x is *covered* by y if $x(i) = 1$ implies $y(i) = 1$, for all indices i. The following definition is due to Dyachkov and Rykov [10].

Definition 3. *[10] Let d, q, n be integers > 0 with $d + q \leq n$ and let $\mathcal{M} = \{M(i, j)\}$ be a binary code of size n. We say that $\mathcal{M}$ is a (d, q)-*superimposed code *if for any $d + q$ codewords $M(\cdot, j_1), \ldots, M(\cdot, j_{d+q})$, there exists a row index i such that $\vee_{h=1}^{d} M(i, j_h) = 1$ and $\vee_{k=d+1}^{d+q} M(i, j_k) = 0$, where $\vee$ denotes boolean OR. The minimal length of a (d, q)-superimposed code of size n is denoted by $N(d, q, n)$.*

The codes in the above definition have the property that the boolean sum of any d columns is not covered by the boolean sum of any other q columns.

For $d = 1$, (d, q)-superimposed codes correspond to classical superimposed codes [13], or equivalently to cover free families [11].

The following asymptotic lower and upper bounds [3, 10] on the minimum length $N(d, q, n)$ of a (d, q)-superimposed code of size n hold:

$$N(d,q,n) = O\left(\frac{q^2}{d}\log\frac{n}{q}\right); \qquad N(d,q,n) = \begin{cases} \Omega\left(\frac{q^2}{d\log q}\log\frac{n}{q}\right) & \text{if } q \geq 2d, \\ \Omega(q\log\frac{n}{d}) & \text{if } q < 2d. \end{cases}$$

The best constructions for (d, q)-superimposed codes achieve the above upper bound with the constant hidden in the big-O notation being smaller than three. We refer the interested readers to [8] for an account on the theory of superimposed codes.

3 Using Group Testing to Perform Corruption Localization

The basic idea of corruption-localizing hash schemes is to define a collection $\mathcal{L}$ of segment lists of the original string x in such a way that the output set T (recall that T is a superset of the indices corresponding to corrupted bits) is determined by comparing the hash tag of each segment list in $\mathcal{L}$ with the hash tag of the corresponding segment list of x', each hash tag being produced using a collision-intractable hash function. Notice that if the hash tag of a segment list of x is equal to the hash tag of the corresponding segment list of x' then no segment in the segment list has been corrupted (unless collisions were found in the collision-intractable hash function), whereas if those two tags are different then at least one segment in the segment list has been corrupted.

Testing whether a collision-resistant hash function maps two segment lists to the same tag corresponds to posing a YES/NO query of the form "Does the tested segment list contain at least one corrupted segment?" thus implying that the problem of determining the corrupted segments in x' by a corruption-localizing hash scheme can be regarded as a particular instance of the group testing problem.

The authors of [5] pointed out that strategies for group testing cannot be used to solve our problem in that those strategies search for atomic elements, whereas we are interested in searching segments that may have lengths that are potentially > 1 and different among each other. One approach to overcome this objection is to focus on corrupted bits rather than corrupted segments. Hence, we state our problem as that of searching for corrupted bits given that these bits occur in at most v segments each of which has length at most β. Having rephrased the problem in these terms, one has to cope with a more specific version of combinatorial group testing where some additional information is known on how positive elements are distributed. Notice that, in order to retrieve the original message from its corrupted version, one has to precompute the hash tag for each segment list of x that belongs to $\mathcal{L}$. In group testing this corresponds to the situation where each query must be decided beforehand without looking at the responses to the previous queries; in other words, a non-adaptive search strategy is needed. The best constructions for $(1, q)$-superimposed codes imply an $O(q^2 \log(n/q))$ upper bound on the cost of non adaptive group testing strategies that search for up to q positive elements. Therefore, determining the exact location of all corrupted bits by a non-adaptive strategy for classical group testing would cost $O((v\beta)^2 \log(n/(v\beta)))$. While this fact alone suffices to obtain a corruption-localizing hash scheme, the bound obtained on the tag length is not satisfactory as it can be linear or super-linear in n whenever $\beta = \Omega(\sqrt{n})$. (Note that in general $\beta = O(n)$ and that our goal was to obtain a corruption-localizing hash scheme with tag length smaller than $o(n)$; e.g. polylogarithmic in n.) Alternatively, one might use a $(d, v\beta)$-superimposed code to obtain a superset of all corrupted bits that contains less than d bits incorrectly classified as corrupted, thus obtaining a localization factor smaller than or equal to $1 + (d-1)/p$, where $p < v\beta$ is the *unknown* exact number of corrupted bits. Since p might be as small as 1 then the localization factor might be as large as d. This localizing strategy would cost $O(((v\beta)^2/d)\log(n/(v\beta)))$, thus achieving a constant localization factor at the same cost incurred by the algorithm that exactly determines all corrupted bits. To improve on the above bound on the tag length, we have to specifically target corruption localization (as opposed to search) and exploit the additional information that corrupted bits occur in segments of length at most β. In Sections 5 and 6 we propose two non-trivial uses of superimposed codes that achieve desirable combinations of efficiency and localization properties.

4 Localizing Codes

In this section we recall the definition of localizing codes [2]. Such codes are used in conjunction with collision-intractable hashing to obtain corruption-localizing hash schemes. Localizing codes are formally defined as binary codes for which there exists an efficient algorithm that, given the output of a matrix product between the code matrix and a corruption vector, returns a superset of the corrupted bits, that is larger by a bounded factor. We use matrix product in the 2-value Boolean algebra; that is, the semiring $(\{0,1\}, \vee, \wedge)$, where matrix product is then a sequence of $\vee$'s (i.e., boolean ORs) of $\wedge$'s (i.e., boolean ANDs). The weight of a binary vector e, defined as the number of nonzero vector components, is denoted as $w(e)$.

Definition 4. *Let $\mathcal{M} = \{M(i,j)\}$ be a binary code of size n and length N. We say that $\mathcal{M}$ is a (β, v, α)-*localizing code* if there exists an efficient algorithm $L_{\mathcal{M}}$ such that for any (β, v)-corruption vector e, given as input the matrix product $M \cdot e$, $L_{\mathcal{M}}$ returns a vector u such that u covers e and $w(u) = \alpha\, \mathsf{Diff}_v[0^n, e]$.*

Corruption Localization from Localizing Codes. The following result [2] shows that any localizing code can be used to construct a corruption-localizing hash scheme from a family of collision-intractable hash functions.

Theorem 1. *Let $\mathcal{H} = \{H_\lambda\}_{\lambda \in \mathcal{N}}$ be a family of hash functions, and let $\mathcal{M} = \{M(i,j)\}$ be a binary code of size n and length N. If $\mathcal{H}$ is (t, ϵ)-collision-intractable and $\mathcal{M}$ is a (β, v, α)-localizing code, then there exists a hash scheme $HS_{\mathcal{M}} = (clH_{\mathcal{M}}, Loc_{\mathcal{M}})$ that is $(t', \epsilon', \beta, v, \alpha')$-corruption-localizing, where $\epsilon' = \epsilon$, $t' = t + O(t_n(H) \cdot N)$. Moreover, $HS_{\mathcal{M}}$ has localization factor $\alpha' = \alpha$, number of hash tags $O(N)$, and runtime complexity $O(t_n(H) \cdot N)$.*

A sketch of the construction of the scheme $HS_{\mathcal{M}}$ is as follows. The hashing algorithm $clH_{\mathcal{M}}$ runs N times function H_λ, where, for $i = 1, \ldots, N$, the i-th execution of H_λ takes as input the concatenation of all file entries $x[j]$ such that $M(i,j) = 1$, for all $j = 1, \ldots, n$, and returns the obtained hash tag. The localizing algorithm $Loc_{\mathcal{M}}$ starts by repeating the same computation by running H_λ on the corrupted file x', and generates an N-bit vector $\mathbf{z}$ such that $\mathbf{z}[i] = 1$ if and only if the hash tag computed on the i-th execution of H_λ on x' is different from the one returned by $clH_{\mathcal{M}}$ as the tag computed by the i-the execution of H_λ on x. Finally, $Loc_{\mathcal{M}}$ runs the algorithm $L_{\mathcal{M}}$ on vector $\mathbf{z}$ to obtain vector u, and outputs this vector. The main observation in the proof of Theorem 1 is that, either collisions are found in H_λ or the vector $\mathbf{z}$ can be shown to be equal to the matrix product between matrix M of the localizing code $\mathcal{M}$ and the (β, v)-corruption vector e, and therefore the localization property of $L_{\mathcal{M}}$ implies an analogue localization property for $Loc_{\mathcal{M}}$.

Theorem 1 unifies previous results (in particular, a result in [6] can be restated as a $(\beta, 1, O(1))$-localizing code, and a result in [5] can be re-stated as a $(\beta, v, O(v^3))$-localizing code) and simplifies the problem of constructing efficient corruption-localizing hash schemes to the problem of constructing efficient localizing codes.

To make the paper self-contained, we report the proof of Thorem 1 in Appendix.

4.1 Using Superimposed Codes to Construct Localizing Codes

In Section 3 we discussed the relationship between group testing and corruption localization. In that section we pointed out that achieving constant localization factor by a trivial application of group testing would cost $O((v\beta)^2 \log n)$ in terms of number of tags, which is not satisfactory as it is at least linear in n whenever $v\beta = \Omega(\sqrt{n/\log n})$. We observed that, to improve on the latter bound on the tag length, we have to specifically target corruption localization (as opposed to search) and exploit the additional information that corrupted bits occur in segments of length at most β. In the next two sections we propose two non-trivial uses of superimposed codes to construct localizing codes with desirable combinations of efficiency and localization properties.

Towards this goal, a crucial idea is to group the bits of x and those of its corrupted version x' into consecutive segments of a fixed number $h \leq \beta$ of bits and to search among these segments for those containing one or more corrupted bits. If the corrupted bits occur in at most a certain number q of these segments, then a group testing strategy for up to q positive elements can be used to compute a solution of size rh, where $r \leq q$ is the number of segments returned by the algorithm. Accordingly, we use a superimposed of size $\lceil n/h \rceil$, whose columns are associated with segments of length h. The main difficulty consists in choosing the integer h so that the corrupted bits can be confined into up to a certain number q of segments of length h, and at the same time it is possible to prove that the total size of these segments is not too far from $\mathsf{Diff}_v[x, x']$. Notice that the upper bound β on the length of the largest corrupted segment might be very loose and not indicative of the real length of corrupted segments.

5 A First Scheme

In this section we present a localizing code that results in a localizing-corruption hash scheme that can be instantiated to achieve a localization factor *arbitrarily close* to β and storage and time complexity $O(v^2 \log(n/(\beta v)))$. This localizing code is used in Section 6 as a building block to design an improved localizing code with localization factor *arbitrarily close* to 1, and storage and time complexity proportional to $O(v(\log v) \log n + v(\log \beta) \log(n/v))$, for any number of corruptions and any corruption length.

An informal discussion. Intuitively, we partition the input message x into consecutive segments of length h, for some $h \leq \beta$, (with the eventual exception of the rightmost segment that might have smaller length). We search among these segments for those containing one or more corrupted bits. Accordingly, we start with a binary code $\mathcal{C}$ of size $\lceil n/h \rceil$, whose columns are associated with these segments. This code is then expanded into a binary code $\mathcal{D}$ of size n by replicating each row entry h times, so that the resulting code essentially operates over bits directly. As already observed in the previous section, it is crucial to choose the integer h so that the corruptions can be confined into up to a certain number of segments of length h, and at the same time the total size of these segments is not too far from $\mathsf{Diff}_v[x, x']$. Towards this goal, a first difficulty is that the lengths of corruptions, although known to be at most β, can actually vary widely between 1 and β. We deal with this difficulty by adding a parameter $b \leq \beta$ to the code, which intuitively denotes a close upper bound on the maximum length of all v corruptions. Accordingly, the input message x is partitioned into segments of some length $h \leq b$. In the following, we set $h = \lceil b/\ell \rceil$, for some $\ell \geq 1$.

The code $\mathcal{D}$ and its properties. Let $\mathcal{C} = \{C(i,j)\}$ be a binary code of size $\lceil n/h \rceil$, with $h = \lceil b/\ell \rceil$ for some positive integer $b \leq \beta$ and an arbitrary constant $\ell \geq 1$, and let N denote the length of $\mathcal{C}$. We define the binary code $\mathcal{D} = \{D(i,j)\}$ of size n and length N, as the code returned by the following procedure.

```
1. For $i = 1, \ldots, N$,
2.     for $j = 1, \ldots, \lceil n/h \rceil$,
3.         let $s_j^b$ denote the interval $[h(j-1), \min\{hj-1, n-1\}]$;
4.         for all $j'$ in interval $s_j^b$,
5.             set $D(i,j') = C(i,j)$.
6. Return: $\mathcal{D} = \{D(i,j)\}$.
```

Code $\mathcal{D}$ satisfies the following theorem.

Theorem 2. *Let b, n, β, v be given positive integers such that $v \leq n$ and $2 \leq b \leq \beta \leq n$, and let $\mathcal{C}$ be a binary code of size $\lceil n/\lceil b/\ell \rceil \rceil$, for some constant $\ell \geq 1$. If $\mathcal{C}$ is a $(d, (\ell+1)v)$-superimposed code, for an arbitrary positive integer $d \leq n - (\ell+1)v$, then $\mathcal{D}$ is a (b, v, α)-localizing binary code of size n, with $\alpha \leq (d+\ell)\lceil b/\ell \rceil$. Moreover, the output vector u has weight $w(u) \leq \lceil b/\ell \rceil((\ell+1)v + d - 1)$.*

Proof. We formally describe an algorithm $L_{\mathcal{D}}$ that localizes up to v corruptions from the (β, v)-corruption vector e, and satisfies the stated bound on the localization factor α. Let $S_1, \ldots, S_v$ be v intervals of length $\leq \beta$ that achieve $\mathsf{Diff}_v[x, x']$. In fact, we prove the stronger inequality $\alpha \leq \lceil b/\ell \rceil((\ell+1) + (d-1)/p)$, where $p \leq v$ is the unknown number of non-empty corrupted segments; that is, p is the number of intervals $S_r \neq \emptyset$.

We define the locator algorithm $L_{\mathcal{D}}$ so that, on input the matrix product $\mathbf{z}$ between the code $\mathcal{D}$'s matrix and a corruption vector e, it returns a set of input indices associated with $\mathbf{z}$ according to a natural vector coverage notion. Then, we prove that $\mathcal{D}$ and $L_{\mathcal{D}}$ satisfy the definition of localizing codes with $\alpha \leq \lceil b/\ell \rceil((\ell+1) + (d-1)/p)$. One important step in proving this fact consists of using the property of superimposed codes to limit the weight of the vector returned by $L_{\mathcal{D}}$. As in the definition of code $\mathcal{D}$, we denote by $h = \lceil b/\ell \rceil$ the length of the segments that the input message is partitioned into.

The algorithm $L_{\mathcal{D}}$. On input the response vector $\mathbf{z} = D \cdot e$, algorithm $L_{\mathcal{D}}$ goes as follows:

```
1. set $u = 0^n$
2. for $j = 1, \ldots, \lceil n/h \rceil$,
3.     let $s^b_j$ denote the interval $[h(j-1), \min\{hj-1, n-1]$;
4.     if column $C(\cdot, j)$ is covered by $\mathbf{z}$ then
5.         set $u[j'] = 1$ for all $j' \in s^b_j$
6. return: $u$
```

We now prove that for any (b, v)-corruption vector e, it holds that $w(u) \leq \alpha\, \mathsf{Diff}_v[0^n, e]$ for $\alpha \leq b + b/\ell + (d-1)b/(\ell p)$. In the following, we say that an interval $s^b_j \in \{s^b_1, \ldots, s^b_{\lceil n/h \rceil}\}$ is *corrupted* if it intersects one or more S_r's.

First we show that the returned vector u has nonzero elements in correspondence of $\leq (\ell{+}1)p{+}d{-}1$ intervals from $\{s^b_1, \ldots, s^b_{\lceil n/h \rceil}\}$, and consequently $w(u) \leq ((\ell{+}1)p{+}d-1)h$. To see this, observe that each S_r has size $\leq b$, and thus each S_r intersects at most $\ell{+}1$ adjacent intervals among $s^b_1, \ldots, s^b_{\lceil n/h \rceil}$. Since there are p intervals S_r's such that $S_r \neq \emptyset$, then there are at most $(\ell+1)p$ corrupted intervals in $\{s^b_1, \ldots, s^b_{\lceil n/h \rceil}\}$. Let $s^b_{j_1}, \ldots, s^b_{j_m}$, for some $m \leq (\ell+1)p$, be the corrupted intervals in $\{s^b_1, \ldots, s^b_{\lceil n/h \rceil}\}$. The response vector $\mathbf{z}$ is the bitwise OR of columns $C(\cdot, j_1), \ldots, C(\cdot, j_m)$. By definition of $(d, (\ell+1)v)$-superimposed code, one has that, for any d column indices $h_1, \ldots, h_d \notin \{j_1, \ldots, j_m\}$, the bitwise OR of columns $C(\cdot, h_1), \ldots, C(\cdot, h_d)$ is not covered by $\mathbf{z}$, and consequently, there might be at most $d-1$ columns, in addition to $C(\cdot, j_1), \ldots, C(\cdot, j_m)$, that are covered by $\mathbf{z}$. Let $C(\cdot, r_1), \ldots, C(\cdot, r_g)$, for some $g < d$, denote these columns. One has that $u[j'] = 0$ if $j' \in s^b_j$ with $j \notin \{j_1, \ldots, j_m, r_1, \ldots, r_g\}$, and $u[j'] = 1$ if $j' \in s^b_j$ with $j \in \{j_1, \ldots, j_m, r_1, \ldots, r_g\}$.

Observe that $\mathsf{Diff}_v[0^n, e]$ might be as small as p since each non-empty interval S_r might consist of a single bit. Hence, $\alpha = w(u)/\mathsf{Diff}_v[0^n, e] = (m+g)h/\mathsf{Diff}_v[x, x'] \leq ((\ell+1)p + d - 1)\lceil b/\ell \rceil$. Since $p \geq 1$, then $\alpha \leq (d + \ell)\lceil b/\ell \rceil$.

The next corollary follows from Theorems 1 and 2.

Corollary 1. *Let b, n, β, v be given positive integers such that $v \leq n$ and $2 \leq b \leq \beta \leq n$, and let $N = N(d, (\ell{+}1)v, \lceil n\ell/b \rceil)$, for an arbitrary constant $\ell \geq 1$ and an arbitrary positive integer $d \leq n - (\ell{+}1)v$. If $\mathcal{H} = \{H_\lambda\}_{\lambda \in \mathcal{N}}$ is a (t, ϵ)-collision-intractable family of hash functions, then there exists a (t', ϵ', b, v)-corruption-localizing hash scheme with $\epsilon' = \epsilon$, $t' = t + O(t_n(H) \cdot N)$, localization factor $\alpha \leq (d + \ell)\lceil b/\ell \rceil$, and number of tags $O(N)$.*

We minimize the localization factor by setting $d = 1$, and apply the upper bound on $N(1, (\ell+1)v, \lceil n\ell/b \rceil)$ so as to obtain a hash scheme with localization factor $(1 + \ell)\lceil b/\ell \rceil$ (*constant* in v) and number of tags $O(v^2 \log(n/(bv)))$(sublinear when $v = o(\sqrt{n/\log n})$), where the hidden constant is smaller than $3(\ell+1)^2$.

6 A Localizing Code with Arbitrarily Small Localization Factor

In this section we present a localization code $\mathcal{E}$ that, when plugged into Theorem 1, results in a corruption-localizing hash scheme with localization factor arbitrarily close to 1, for any number of corruptions and any corruption length.

An informal description. The basic idea behind our construction is that we need to learn a good estimate b of the length of the largest corrupted segment. As already observed, this parameter highly affects the efficiency and localization capabilities of localizing codes. With respect to the scheme in Section 5, we observe that if b is chosen too large then upper bound $(d+\ell)\lceil b/\ell \rceil$ on the localization factor achieved by the scheme is also large. On the other hand, if the chosen value is too small then the number of corrupted segments might be larger than the parameter $q=(\ell+1)v$ of the superimposed code $\mathcal{C}$, and consequently, the number of columns of $\mathcal{C}$ covered by the response vector might be much larger than $(\ell+1)v$, in which case $w(u)$ would be very large as well. In order to learn a good estimate of the length of the largest corrupted segment, we apply the scheme of Section 5 with decreasing values of b, ranging from β to 1.

We construct the localizing code $\mathcal{E}$ and the related localizing algorithm $L_{\mathcal{E}}$ as follows. Let $q>1$ and $\ell \geq 1$ be arbitrarily chosen constants. For $k=0,\ldots,\lceil \log_q \beta \rceil$, we denote by $\mathcal{C}_k$ a binary code of size $\lceil n/h_k \rceil$, where $\beta_k = \lceil \beta/q^k \rceil$ and $h_k = \lceil \beta_k/\ell \rceil$. Code $\mathcal{C}_k$ is expanded into a binary code $\mathcal{D}_k$ by replicating each row entry h_k times. Notice that code $\mathcal{D}_k$ corresponds to code $\mathcal{D}$ of Section 5 with the parameter b set equal to β_k. Intuitively, $\mathcal{D}_k$ obtains a good localization factor if the maximum length of a corrupted segment ranges in the interval $(\lceil \beta/q^{k+1} \rceil, \lceil \beta/q^k \rceil]$.

Given codes $\mathcal{C}_k$ and $\mathcal{D}_k$, we construct a localizing algorithm L_k as in the proof of Theorem 2, and use that theorem to conclude that the n-bit vector u_k returned by L_k satisfies the following properties: (a) u_k covers the (β,v)-corruption vector e, and (b) if all corrupted segments have length $\leq \beta_k = \lceil \beta/q^k \rceil$ then $w(u_k) \leq \lceil \beta_k \ell^{-1} \rceil ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)$. The idea of our localizing algorithm $L_{\mathcal{E}}$ is to run algorithms $L_0,\ldots,L_f$, where $f = \min\{k \in \{1,\ldots,\lceil \log_q \beta \rceil\} : w(u_k) > \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)\}$. A crucial difficulty consists in proving that, by stopping at $k=f$, the algorithm $L_{\mathcal{E}}$ achieves the desired localization factor. It is also possible that the localizing algorithm never finds such an f, in which case it is not possible to prove that one of vector u_k's attains the desired localization factor. In order to achieve the desired localization factor, we exploit the fact that, in this case, the total number of corrupted bits is at most $w(u_{\lceil \log \beta \rceil}) \leq \ell^{-1}((\ell+1)v + \lceil v\ell/c \rceil + d - 1)$, and consequently, it is convenient to localize all corruptions by a group testing strategy that directly searches for the single corrupted bits, rather than searching for corrupted intervals of some maximum length. Observe that a trivial use of non-adaptive group testing would allow to exactly detect the corrupted bits but would incur an overhead of $O((v+d)^2 \log(n/(v+d)))$. Alternatively, if we knew a good estimate of the exact number of corrupted bits then we would be able to build a localizing code with a small localization factor. Hence, we need to guess a good estimate of the number of corrupted bits and, to this aim, we add a parameter v_r to the localizing code, which intuitively denotes a close upper bound on the number of corrupted bits. We set $v_0 = (\ell+1)v + \lceil v\ell/c \rceil + d - 1$, that is v_0 is the known upper bound on the number of corrupted bits, and $v_r = \lfloor v_0/m^r \rfloor$ for an arbitrary constant $m>1$. Notice that, for some integer $r \in [0, \lfloor \log_m v_0 \rfloor]$, v_r is a close upper bound on the exact number of corrupted bits. For $r=0,\ldots,\lfloor \log_m v_0 \rfloor$, we denote by $\mathcal{G}_r$ a localizing code that localizes up to v_r corrupted bits. The code $\mathcal{E}$ is obtained by concatenating the rows of codes $\mathcal{D}_k$'s and $\mathcal{G}_r$'s.

The code $\mathcal{E}$ and its properties. Let $c > 0$, $\ell \geq 1$, $q > 1$, $m > 1$ be arbitrarily chosen constants. For $k = 0, \ldots, \lceil \log_q \beta \rceil$, let $\beta_k = \lceil \beta/q^k \rceil$ and $h_k = \lceil \beta_k/\ell \rceil$, and let $\mathcal{C}_k$ be a binary code of size $\lceil n/h_k \rceil$ whose length is denoted by N_k. Moreover, for $r = 0, \ldots, \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor$, let $\mathcal{G}_r$ denote a binary code of size n whose length is denoted by $\hat{N}_r$. Code $\mathcal{E}$ is formally defined as follows:

```
 1. Set $h = 0$
 2. For $k = 0, \ldots, \lceil \log_q \beta \rceil$,
 3.     for $i = 1, \ldots, N_k$,
 4.         for $j = 1, \ldots, \lceil n/\beta_k \rceil$,
 5.             let $s_j^{\beta_k}$ denote the interval $[\lceil \beta_k \ell^{-1} \rceil (j-1), \min\{\lceil \beta_k \ell^{-1} \rceil j - 1, n-1\}]$,
 6.             for all $j'$ in interval $s_j^{\beta_k}$,
 7.                 set $D_k(i, j') = C_k(i, j)$ and $E(h+i, j') = D_k(i, j')$
 8.     set $h = h + N_k$
 9. For $r = \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor, \ldots, 0$,
10.     for $i = 1, \ldots, \hat{N}_r$,
11.         for $j = 1, \ldots, n$,
12.             set $E(h+i, j) = G_r(i, j)$
13.     $h = h + \hat{N}_r$.
14. Return: $\mathcal{E} = \{E(i, j)\}$.
```

If codes $\mathcal{C}_k$'s are $(d, (\ell+1)v + \lceil v\ell/c \rceil)$-superimposed codes of size $\lceil n/h_k \rceil$ for an arbitrary $d \leq n - (\ell+1)v - \lceil v\ell/c \rceil$, and codes $\mathcal{G}_r$'s are $(\lfloor v_r/c' \rfloor, v_r)$-superimposed codes of size n, with $v_r = \lfloor ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)/m^r \rfloor$ and $c' \geq 1$ being an arbitrary constant, then it is possible to prove that code $\mathcal{E}$ achieves localization factor $\alpha \leq \max\{q(1 + c + c/\ell + c(d-1)/(v\ell)), m + m/c'\}$. The number of tags of the related scheme is the sum of the lengths of $\mathcal{C}_0, \ldots, \mathcal{C}_{\lceil \log \beta \rceil}, \mathcal{G}_0, \ldots, \mathcal{G}_{\lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor}$. This leads to the main results of the paper .

Theorem 3. *Let n, β, v be given positive integers such that $\beta \leq n$ and $v \leq n$, and let $c > 0$, $c' \geq 1$, $\ell \geq 1$, $m > 1$, $q > 1$ be arbitrarily chosen constants. Moreover, let $N = \sum_{k=0}^{\lfloor \log_q \beta \rfloor} N(d, (\ell+1)v + \lceil v\ell/c \rceil, \lceil n/\lceil \beta_k/\ell \rceil \rceil) + \sum_{r=0}^{\lfloor \log_m v_0 \rfloor} N(\lfloor v_r/c' \rfloor, v_r, n)$, with $\beta_k = \lceil \beta/q^k \rceil$, $v_r = \lfloor ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)/m^r \rfloor$, and with $d \leq n - (\ell+1)v - \lceil v/c \rceil$ being an arbitrary positive integer. If $\mathcal{H} = \{H_\lambda\}_{\lambda \in \mathcal{N}}$ is a (t, ϵ)-collision-intractable family of hash functions then there exists a $(t', \epsilon', \beta, v)$-corruption-localizing hash scheme, with $\epsilon' = \epsilon$, $t' = t + O(t_n(H) \cdot N)$, localization factor $\alpha \leq \max\{q(1 + c + c/\ell + c(d-1)/(v\ell)), m + m/c'\}$ and number of tags $O(N)$.*

Observe that the upper bound on the minimum length of (d, q)-superimposed codes implies that the minimum length of a $(d, (\ell+1)v + \lceil v/c \rceil)$-superimposed code of size $\lceil n/\lceil \beta_k/\ell \rceil \rceil$ is $O((v^2/d) \log(nq^k/(\beta v)))$, where the hidden constant is smaller than $3(\ell+1)^2$. Moreover, by the same bound, the minimum length of a $(\lfloor v_r/c' \rfloor, v_r)$-superimposed code of size n is $O(v_r \log(n/v_r)$, where the hidden constant is smaller than $3c'$. These bounds imply that the number of hash tags of the scheme of Theorem 3 is upper bounded by $O((v+d) \log n + (v^2/d)(\log \beta) \log(n/v)))$, with the hidden constant being $< \max\{3(\ell+1)^2/\log q, 3c'(\ell + 1 + \ell/c)/\log m\}$.

When setting $d = v$ in Theorem 3, the scheme achieves localization factor $\alpha \leq \max\{q(1 + c + 2c/\ell), m + m/c'\}$ and number of tags $O(v \log n + v \log \beta \log(n/v))$. Therefore, we have that the following result holds.

Corollary 2. *Let n, β, v be given positive integers such that $\beta \le n$ and $v \le n$. There exists a corruption-localizing hash scheme with number of tags $O(v \log n + v \log \beta \log(n/v))$ and localization factor $\alpha \le \max\{q(1+c+2c/\ell), m+m/c'\}$, where $m > 1$, $q > 1$, $c > 0$ and $c' \ge 1$ are arbitrary constants.*

We remark that the constant $c > 0$, $c' \ge 1$, $\ell \ge 1$, $m > 1$ and $q > 1$ can be chosen arbitrarily so as to obtain a localization factor arbitrarily close to 1.

Proof of Theorem 3. We formally describe an algorithm $L_{\mathcal{E}}$ that localizes up to v corruptions. We denote by e the (β, v)-corruption vector that transforms x into x'. For $k = 0, \ldots, \lceil \log_q \beta \rceil$, $\mathcal{C}_k$ and $\mathcal{D}_k$ are the binary codes used to define the first rows of code $\mathcal{E}$, and L_k denotes the algorithm obtained by replacing $\mathcal{C}$ with $\mathcal{C}_k$ and $\mathcal{D}$ with $\mathcal{D}_k$ in the algorithm $L_{\mathcal{D}}$ described in Section 4. Moreover, for $r = 0, \ldots, \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor$, $\mathcal{G}_r$ is the binary code used to define the last rows of code $\mathcal{E}$.

The algorithm $L_{\mathcal{E}}$. On input the response vector $\mathbf{z} = \mathcal{E} \cdot e$, $L_{\mathcal{E}}$ does the following:

```
 1. Set $u_k = 0^n$, for $k = 0, \ldots, \lceil \log_q \beta \rceil$.
 2. For $k = 0, \ldots, \lceil \log_q \beta \rceil$,
 3.     set $u_k =$ the vector returned by $L_k$,
 4.     if $w(u_k) > \lceil \beta_k \ell^{-1} \rceil ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)$,
 5.         then return: $u = u_{k-1}$,
 6.     set $k = k + 1$.
 7. Set $\hat{u}_r = 0^n$, for $r = 0, \ldots, \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor$.
 8. For $r = \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor, \ldots, 0$,
 9.     for $j = 1, \ldots, n$,
10.         if column $G_r(\cdot, j)$ is covered by $\mathbf{z}$
11.             then set $\hat{u}_r[j] = 1$
12.     if $w(\hat{u}_r) < v_r + \lfloor v_r/c' \rfloor$
13.         then return: $u = \hat{u}_r$
14.     set $r = r - 1$.
```

It is immediate to see that the vector u returned by $L_{\mathcal{E}}$ covers the corruption vector e. To prove Theorem 3, we need to show that, by an appropriate choice of codes $\mathcal{C}_k$'s and G_r's, the vector u achieves the claimed upper bound on α. The proof of Theorem 3 relies on the following two lemmas.

Lemma 1. *Let n, β, v be given positive integers such that $\beta \le n$ and $v \le n$, and let $c > 0$, $\ell \ge 1$, $q > 1$ be arbitrarily chosen constants. For $k = 0, \ldots, \lceil \log_q \beta \rceil$, let $\beta_k = \lceil \beta / q^k \rceil$, and let $\mathcal{C}_k$ be a $(d, (\ell+1)v + \lceil v\ell/c \rceil)$-superimposed code of size $\lceil n / \lceil \beta_k/\ell \rceil \rceil$, where $d \le n - (\ell+1)v - \lceil v\ell/c \rceil$ is an arbitrary positive integer. If algorithm $L_{\mathcal{E}}$ terminates at line 5, then it achieves localization factor $\alpha < q(1 + c + c/\ell + c(d-1)/(v\ell))$.*

Proof. For the sake of simplicity, we will assume that $v\ell/c$ is an integer (the more general case being proved similarly).

Algorithm $L_{\mathcal{E}}$ terminates at line 5 if and only if there is an integer $k \in [0, \lceil \log_q \beta \rceil]$ such that $w(u_k) > \lceil \beta_k \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$. In this case algorithm $L_{\mathcal{E}}$ returns the vector u_{f-1}, where $f = \min\{k \in \{1, \ldots, \lceil \log_q \beta \rceil\} : w(u_k) > \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)\}$. Let $S_1, \ldots, S_v$ be v intervals that achieve $\mathsf{Diff}_v[x, x']$. Theorem 2 implies that there exists an $h \in \{0, \dots, n-1\}$ such that $|S_h| > \beta_f = \lceil \beta q^{-f} \rceil$. Indeed, any $(d, (\ell+1)v + v\ell/c)$-superimposed code is also a $(d, (\ell+1)v)$-superimposed code and consequently $\mathcal{C}_f$ satisfies the hypothesis of Theorem 2. By that theorem, one has that if $|S_r| \leq \beta_f$, for all $r = 1, \dots, v$, then $w(u_f) \leq \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + d - 1)$. Since we have chosen f as the smallest integer such that $w(u_f) > \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$ then at least one of $S_1, \dots, S_v$ should have cardinality larger than β_f.

Now we show that $\mathsf{Diff}_v[x, x'] > \beta q^{-f} v/c$. Let us assume by contradiction that $\mathsf{Diff}_v[x, x'] \leq \beta q^{-f} v/c$ and let us consider the partition of $[0, n-1]$ into the intervals $s_1^{\beta_f}, \dots, s_{\lceil n/\lceil \beta_f/\ell \rceil \rceil}^{\beta_f}$ of length $\lceil \beta_f/\ell \rceil$ (with the eventual exception of the rightmost interval that might have smaller length). For $r = 1, \dots, v$, we denote by $I_r^{\beta_f}$ the set of (consecutive) corrupted intervals $s_j^{\beta_f}$'s that intersect S_r. Notice that for some $r < v$, $I_r^{\beta_f}$ and $I_{r+1}^{\beta_f}$ might intersect since the rightmost corrupted interval in $I_r^{\beta_f}$ might be the leftmost in $I_{r+1}^{\beta_f}$. Let $I_{h_1}^{\beta_f}, \dots, I_{h_g}^{\beta_f}$, for some $g \leq v$, denote those sets among $I_1^{\beta_f}, \dots, I_v^{\beta_f}$ with cardinality larger than or equal to $\ell + 2$. We have that $\mathsf{Diff}_v[x, x'] = \sum_{r=1}^{v} |S_r| \geq \sum_{j=1}^{g} |S_{h_j}| > \sum_{j=1}^{g} (|I_{h_j}^{\beta_f}| - 2) \lceil \beta_f \ell^{-1} \rceil$. Indeed, for $j = 1, \dots, g$, S_{h_j} entirely contains all intervals in $I_{h_j}^{\beta_f}$, with the eventual exception of the leftmost and rightmost segments that are not taken into account by the summation in the last term of the above inequality. This inequality and the contradiction hypothesis imply that $\sum_{j=1}^{g} (|I_{h_j}^{\beta_f}| - 2) \lceil \beta_f \ell^{-1} \rceil < \beta_f v/c$ from which it follows that $\sum_{j=1}^{g} |I_{h_j}^{\beta_f}| < v\ell/c + 2g$. On the other hand the number of corrupted intervals $s_j^{\beta_f}$'s is at most $\sum_{r=1}^{v} |I_r^{\beta_f}| \leq \sum_{j=1}^{g} |I_{h_j}^{\beta_f}| + (\ell+1)(v-g)$, in view of the fact that $|I_r^{\beta_f}| \leq \ell + 1$ for $r \notin \{h_1, \dots, h_g\}$. The last two inequalities imply that the number of corrupted intervals $s_j^{\beta_f}$'s is at most $(\ell+1)v + v\ell/c$, and consequently, the response vector $\mathbf{z}_f = D_f \cdot e$ is the boolean sum of up to $(\ell+1)v + v\ell/c$ columns of $\mathcal{C}_f$. Since $\mathcal{C}_f$ is a $(d, (\ell+1)v + v\ell/c)$ superimposed code then the only columns covered by $\mathbf{z}_f$ are those associated with the corrupted intervals plus an additional subset of at most $d - 1$ columns. It follows that $w(u_f) \leq \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$, from which we have a contradiction.

We are ready to show that $w(u_{f-1})/\mathsf{Diff}_v[x, x'] < q(1 + c + c/\ell + c(d-1)/(v\ell))$. To see this, observe that f is the smallest index such that $w(u_f) > \lceil \beta_f \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$ and consequently it must hold $w(u_{f-1}) \leq \lceil \beta_{f-1} \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$. On the other hand, we proved that $\mathsf{Diff}_v[x, x'] > \beta q^{-f} v/c$ from which we have that $w(u_{f-1})/\mathsf{Diff}_v[x, x'] < q(1 + c + c/\ell + c(d-1)/(v\ell))$. $\square$

Lemma 2. *Let n, β, v be given positive integers such that $\beta \leq n$ and $v \leq n$, and let $c > 0$, $c' \geq 1$, $\ell \geq 1$, $m > 1$, $q > 1$ being arbitrarily chosen constants. For $k = 0, \dots, \lceil \log_q \beta \rceil$, let β_k and $\mathcal{C}_k$ be defined as in the statement of Lemma 1, and for $r = 0, \dots, \lfloor \log_m((\ell+1)v + \lceil v\ell/c \rceil + d - 1) \rfloor$, let $v_r = \lfloor ((\ell+1)v + \lceil v\ell/c \rceil + d - 1)/m^r \rfloor$ and $\mathcal{G}_r$ be a $(\lfloor v_r/c' \rfloor, v_r)$-superimposed code of size n. If algorithm $L_{\mathcal{E}}$ terminates at line 13, then it achieves localization factor $\alpha \leq m + m/c'$.*

Proof. Algorithm $L_{\mathcal{E}}$ terminates at line 13 only if $w(u_k) \leq \lceil \beta_k \ell^{-1} \rceil ((\ell+1)v + v\ell/c + d - 1)$, for all $k = 0, \dots, \lceil \log_q \beta \rceil$. In this case the algorithm outputs the vector $\hat{u}_g$, with $g = \max\{r \in \{0, \dots, \lfloor \log_m v_0 \rfloor\} : w(\hat{u}_k) \leq v_r + \lfloor v_r/c' \rfloor\}$. We will show that $w(\hat{u}_k)/\mathsf{Diff}_v[x, x'] < m(1 + 1/c')$. Let $p \leq v$ denote the unknown number of corrupted bits. Since $p \leq w(u_k)$, for $k = 0, \ldots, \lceil \log_q \beta \rceil$, and $\beta_{\lceil \log_q \beta \rceil} = 1$, then $p \leq (\ell + 1)v + v\ell/c + d - 1$. Let us define $v_0 = (\ell + 1)v + \lceil v\ell/c \rceil + d - 1$. We denote by a the integer in $\{0, \ldots, \lfloor \log_m v_0 \rfloor\}$ such that $v_0/m^{a+1} < p \leq v_0/m^a$. Let $G_a(\cdot, j_1), \ldots, G_a(\cdot, j_p)$ be the p columns of G_a associated with the p corrupted bits. The response vector $\hat{\mathbf{z}}_a = G_a \cdot e$ is the bitwise OR of columns $G_a(\cdot, j_1), \ldots, G_a(\cdot, j_p)$. By definition of $(\lceil v_a/c' \rceil, v_a)$-superimposed code, one has that the bitwise OR of any $\lceil v_a/c' \rceil$ columns is not covered by the union of up to v_a other columns. Therefore, for any $\lceil v_a/c' \rceil$ column indices $h_1, \ldots, h_{\lceil v_a/c' \rceil} \notin \{j_1, \ldots, j_p\}$, the bitwise OR of columns $G_a(\cdot, h_1), \ldots, G_a(\cdot, h_{\lceil v_a/c' \rceil})$ is not covered by $\hat{\mathbf{z}}$, and consequently, there might be at most $\lceil v_a/c' \rceil - 1$ columns, in addition to $G_a(\cdot, j_1), \ldots, G_a(\cdot, j_p)$, that are covered by $\mathbf{z}$. This guarantees, that algorithm $L_{\mathcal{E}}$ will return a vector $\hat{u} = \hat{u}_g$ (for some $g \geq a$) such that $w(\hat{u}_g) < v_g + \lceil v_g/c' \rceil \leq v_a + \lceil v_a/c' \rceil$, and therefore, the algorithm achieves localization factor $\alpha = w(u)/\mathsf{Diff}_v[0^n, e] < (v_a + \lfloor v_a/c' \rfloor)/p$. Since $p > v_0/m^{a+1} = v_a/m$, it follows that $\alpha < (mp + mp/c')/p = m(1 + 1/c')$. □

Lemmas 1 and 2 imply that $\mathcal{E}$ is a (β, v, α)-localizing code with $\alpha \leq \max\{q(1 + c + c/\ell + c(d-1)/(v\ell)), m + m/c'\}$. Therefore, Theorem 3 is a consequence of Theorem 1 and of the above two lemmas.

References

1. I. Damgard, “Collision Free Hash Functions and Public Key Signature Schemes”, in *Advances in Cryptology - EUROCRYPT’ 87*, pages 203-216, LNCS, Springer-Verlag.
2. A. De Bonis, G. Di Crescenzo, “Combinatorial Group Testing for Corruption Localizing Hashing”, to appear in *Proceedings of The 17th Annual International Computing and Combinatorics Conference - COCOON’11*, LNCS, Springer Verlag.
3. A. De Bonis, L. Gasieniec and U. Vaccaro, “Optimal Two-Stage Algorithms for Group Testing Problems”, *SIAM Journal on Computing*, vol. **34**, No. 5, pp. 1253-1270, 2005.
4. G. Di Crescenzo, R. Ge and G. Arce, “Design and Analysis of DBMAC: an Error-Localizing Message Authentication Code”, *Proceedings of IEEE GLOBECOM ’04*.
5. G. Di Crescenzo, S. Jiang, and R. Safavi-Naini, “Corruption-localizing hashing”, *Proceedings of Computer Security - ESORICS 2009*, LNCS **5789**, Springer Verlag, 489-504, 2009.
6. G. Di Crescenzo and F. Vakil, “Cryptographic hashing for virus localization”, *Proceedings of the 2006 ACM CCS Workshop on Rapid Malcode - WORM’06*, 41-48, 2006.
7. R. Dorfman, “The detection of defective members of large populations”, *Ann. Math. Statist.*, **14**, 436–440, 1943.
8. D.Z. Du and F.K. Hwang, *Pooling Designs and Nonadaptive Group Testing*, World Scientific, 2006.
9. D.Z. Du and F.K. Hwang, *Combinatorial Group Testing and its Applications*, World Scientific, 2000.
10. A.G. Dyachkov, V.V. Rykov, “A survey of superimposed code theory”, *Problems Control & Inform. Theory*, **12**, No. 4, 1–13, 1983.
11. P. Erdös, P. Frankl, and Z. Füredi, “Families of finite sets in which no set is covered by the union of *r* others”, *Israel J. of Math.*, **51**, 75–89, 1985.
12. M. Goodrich, M. Atallah, and R. Tamassia, “Indexing information for data forensics”, *Applied Cryptography and Network Security Conference (ACNS 2005)*, 206–221, 2005.
13. W.H. Kautz and R.R. Singleton, “Nonrandom binary superimposed codes”, *IEEE Trans. on Inform. Theory*, **10**, 363–377, 1964.
14. A. Russell, “Necessary and Sufficient Conditions for Collision-Free Hashing”, in *Journal of Cryptology*, vol. 8, n.2, 1995.

Proof of Theorem 1

We start the proof with a formal description of scheme $HS_{\mathcal{M}} = (clH_{\mathcal{M}}, Loc_{\mathcal{M}})$ and then prove the claimed values on localization factor, number of tags and runtime complexity. The formal description of scheme $HS_{\mathcal{M}}$ can be found in Figure 1 (here, the symbol | denotes string concatenation and $desc(h_\lambda)$ denotes the description of h_λ).

```
Algorithm $clH_{\mathcal{M}}(x)$
1.  randomly choose $h_\lambda$ from $H_\lambda$
2.  for $i \leftarrow 1$ to $N$
3.      $L_i \leftarrow$ empty string
4.      for $j \leftarrow 1$ to $n$
5.          if $M(i,j) = 1$
6.            then $L_i \leftarrow L_i \,|\, x[j]$
7.      $tag_i \leftarrow h_\lambda(L_i)$
8.  $tag \leftarrow (tag_1, \ldots, tag_N, desc(h_\lambda))$
9.  return $tag$

Algorithm $Loc_{\mathcal{M}}(x', tag)$
1.  for $i \leftarrow 1$ to $N$
2.      $L'_i \leftarrow$ empty string
3.      for $j \leftarrow 1$ to $n$
4.          if $M(i,j) = 1$
5.            then $L_i \leftarrow L_i \,|\, x[j]$
6.      $tag'_i \leftarrow h_\lambda(L'_i)$
7.      if $tag_i = tag'_i$
8.        then $\mathbf{z}(i) \leftarrow 0$
9.        else $\mathbf{z}(i) \leftarrow 1$
10. $\mathbf{z} \leftarrow (\mathbf{z}(1), \ldots, \mathbf{z}(N))$
11. $u \leftarrow L_{\mathcal{M}}(\mathbf{z})$
12. return $u$
```

Fig. 1: Our corruption localizing hash scheme $HS_{\mathcal{M}}$.

We observe that the storage complexity and runtime complexity of $HS_{\mathcal{M}}$ claimed in the theorem can be checked by inspection, and concentrate on proving the collision intractability and corruption localization property, and, in particular, the values for ϵ', t', α' claimed in the theorem. We start the rest of the proof by rewriting the probabilities Prob$[\,\mathsf{Succ}_1(A; HS_{\mathcal{M}}; \alpha', v) = 1\,]$ and Prob$[\,\mathsf{Succ}_2(A; HS_{\mathcal{M}}; \alpha', v) = 1\,]$, and then show that both quantities are smaller than $\epsilon' = \epsilon$ by proving that (1) an adversary preventing effective localization (i.e., an index of a modified block is not included in u) can be used to violate the collision intractability of family $\mathcal{H}$; (2) the adversary cannot force the scheme to exceed the localization factor α stated in the theorem.

Let $\mathsf{Coll}(A)$ be the event defined as follows: "For some $i \in \{1, \ldots, N\}$, it holds that $L_i \neq L'_i$ and $h_\lambda(L_i) = h_\lambda(L'_i)$, where L_i, L'_i are two segment lists obtained when running $CLH_{\mathcal{M}}$ on input x, x' returned by A, respectively." We can then write

$$\begin{aligned}
&\text{Prob}[\,\mathsf{Succ}_1(A; HS_{\mathcal{M}}; \alpha', v) = 1\,] \le \text{Prob}[\,\mathsf{Coll}(A)\,], \text{ and}\\
&\text{Prob}[\,\mathsf{Succ}_2(A; HS_{\mathcal{M}}; \alpha', v) = 1\,] \le \text{Prob}[\,\mathsf{Coll}(A)\,] + \\
&\qquad\qquad\qquad \text{Prob}[\,\mathsf{Succ}_2(A; HS_{\mathcal{M}}; \alpha', v) = 1 | \overline{\mathsf{Coll}(A)}\,],
\end{aligned}$$

and then analyze the two probabilities Prob$[\,\mathsf{Coll}(A)\,]$ and Prob$[\,\mathsf{Succ}_2(A; HS_{\mathcal{M}}; \alpha', v) = 1 | \overline{\mathsf{Coll}(A)}\,]$ via the following two lemmas. The first lemma also proves the value in the theorem for the claimed t' and one term of the value for the claimed ϵ' by computing an upper bound on Prob$[\,\mathsf{Coll}(A)\,]$, as follows.

Lemma 3. *If the family $\mathcal{H}$ of hash functions is (t, ϵ)-collision-resistant, then for any algorithm A running in time t', it holds that*

$$\text{Prob}[\,\mathsf{Coll}(A)\,] \le \epsilon,$$

where $t' = t + O(N \cdot t_n(H))$.

Proof. We use algorithm A to obtain an algorithm A' such that, if event $\mathsf{Coll}(A)$ happens then A' violates the collision-intractability of $\mathcal{H}$. On input a hash function h_λ randomly chosen from H_λ, algorithm A', runs the following steps:

```
1. Let $(x, x') = A(1^\lambda)$
2. For $i = 1, \ldots, N$,
     let $L_i =$ empty string and $L'_i =$ empty string;
     for $j = 1, \ldots, n$,
        if $M(i, j) = 1$ then let $L_i = L_i \,|\, x[j]$ and $L'_i = L'_i \,|\, x'[j]$;
     set $tag_i = h_\lambda(L_i)$ and $tag'_i = h_\lambda(L'_i)$;
     if $tag_i = tag'_i$ then return: $(L_i, L'_i)$.
3. return: $\perp$.
```

We see that A' obtains A's output and then runs the same steps as A on input x (resp., x') to obtain strings L_i (resp., L'_i). Thus, we obtain that $\epsilon' = \epsilon$ and $t' = t + O(N \cdot t_n(H))$, where $t_n(H)$ is the max running time of any function H_λ on n-block inputs. □

The second lemma proves the value claimed for α' and one term for the value claimed for ϵ' by proving that when he finds no collisions in H_λ, the adversary succeeds with probability 0 for the value of α stated in the theorem.

Lemma 4. *If $\mathcal{M}$ is a (β, v, α)-localizing code and $\alpha' = \alpha$, for any algorithm A, it holds that*

$$\text{Prob}[\,\mathsf{Succ}_2(A; HS_{\mathcal{M}}; \alpha', v) = 1 | \overline{\mathsf{Coll}(A)}\,] = 0.$$

Proof. If event $\mathsf{Coll}(A)$ does not happen, for all $i \in \{1, \ldots, N\}$, the condition $L_i \ne L'_i$ implies $H_\lambda(L_i) \ne H_\lambda(L'_i)$, where L_i, L'_i are two strings obtained when running $clH_{\mathcal{M}}$ on input x, x' returned by A, respectively. In this case algorithm $Loc_{\mathcal{M}}$ does not add any block to the output u in corresponding of any equality $H_\lambda(L_i) = H_\lambda(L'_i)$. This implies that for $i = 1, \ldots, w - 1$, it holds that $x[j] = x'[j]$ for all j such that $u(j) = 0$.

Thus, the only way that A can make Prob$[\,\mathsf{Succ}_2(A;\mathrm{HS};\alpha,v) = 1|\overline{\mathsf{Coll}(A)}\,] > 0$ is by forcing the locator $Loc_{\mathcal{M}}$'s output u to satisfy a larger localization factor than the α' claimed in the lemma. In the rest of the proof, we show that this is not possible for any algorithm A; that is, the localization factor α' is equal to the parameter α of the (β, v, α)-localizing code.

We see that algorithm $Loc_{\mathcal{M}}$ returns the n-bit vector u that is output by algorithm $L_{\mathcal{M}}$ on input the N-bit vector $\mathbf{z}$. Then we observe that, by construction of $Loc_{\mathcal{M}}$, each component $\mathbf{z}[i]$ of vector $\mathbf{z}$ is equal to 1 if $tag_i \neq tag'_i$, which always happens when $H_\lambda(L_i) \neq H_\lambda(L'_i)$ (as we are assuming that $\mathsf{Coll}(A)$ is false). Now, the inequality $H_\lambda(L_i) \neq H_\lambda(L'_i)$ only holds when $L_i \neq L'_i$, which implies that the j-th bit of vector e is equal to 1 for at least one value j such that $M(i,j) = 1$. Thus, we have that $\mathbf{z}$ can be written as $M \cdot e$, this matrix product being over the semiring $(\{0,1\}, \vee, \wedge)$. We can then apply the property of localizing code $\mathcal{M}$, as from Definition 4, which implies that $w(u) \leq \alpha\, \mathsf{Diff}_v[0^n, e]$, which proves the lemma. □

Given these two lemmas, the theorem then follows by further observing that algorithm $Loc_{\mathcal{M}}$ in $\mathrm{HS}_{\mathcal{M}}$ adds to the output T all but those blocks that appear in some segment L_i for which it holds $h_\lambda(L_i) = h_\lambda(L'_i)$. Therefore, if event $\mathsf{Coll}(A)$ does not happen then the output T contains all corrupted blocks, which implies that if event $\mathsf{Coll}(A)$ does not happen then, for $i = 1, \ldots, w-1$, it holds that $x[\overline{T}] = x'[\overline{T}]$.