CHAPTER 5

SDD FOR DATA RECONSTRUCTIONS

5.1 INTRODUCTION

Hard Decision Decoding (HDD) algorithms such as the Berlekamp-Massey and the Euclidian algorithms are efficient in correcting random and burst errors [40]. As far as storage is concerned, incorrect or corrupt fragment byte sets (from servers with hardware errors or servers under the control of adversary) characterize random and burst errors. Server selections are predominantly handled objectively (integrity verification of data shares using cryptographic tools and techniques) at DSS. Accordingly decoding at DSS-D is extensively HDD, conventionally treating all fragment shares equally. DSS-Ds operate across public networks, facing challenges related to potentially malicious behavior and therefore it is reasonable to expect varying levels of reliability among the storage servers. Belief (confidence) in storage server – the extent to which it is reliable in fragment share submission for data retrieval – should definitely vary from one server to another. It is in fact fitting to deduce that in the presence of malicious activities at storage,

- servers – malicious and non-malicious – at storage should not be equally trusted
- ‘equality in trust’ is certainly not practical

Given that it is practical to assume that storage servers are not to be equally trusted, it is only appropriate to give due importance to the associated trust / reliability values (indices) at decoding. We know that HDD treats all received symbols equally with respect to reliability. Exercising HDD during data retrievals compels equal treatment of storage servers and their fragment shares in reconstructions (decoding). In contrast to HDD, iterative decoding algorithms (class of Chase algorithms (C-1, C-2, and C-3)) [40], Ordered Statistics Decoding
(OSD) algorithm, Generalized Minimum Distance (GMD) decoding algorithm) [16, 44] use reliability information to compute the most likely correct symbol set that has been sent over a noisy channel. This class of algorithms falls under the category of Soft Decision Decoding. SDD algorithms can be most suitably applied to perform iterative decoding during data retrievals in DSS-D.

An interesting case for exploring the use of SDD in DSS-D opens up with storages invariably operating in malicious environments. Interestingly, subjective approach to server selections evidently makes room for the application of iterative decoding algorithms at data retrievals in DSS-D. The scope for a judicious, reliability based selection of storage servers in DSS-D is justified. A subjective ‘trust-reputation’ based judicious selection of reliable storage servers – on a measure of server reliability / trust itself – was introduced in the previous chapter. An organized approach to compute and administer ‘server trust’, as a measure of reliability of servers in terms of fragment share submissions, effectively across data retrievals was developed.

We present the application of SDD in RS coded storages in this chapter; in effectively using reliability (‘server trust’) information for selecting and using the shares for data reconstructions. It is important to note that SDD has not received due deliberation in DSS-D and is in focus in the work presented here. Decoding is primarily ‘hard’. An overview of the decoding problem and the manner in which HDD poses the problem of decoding are presented in Section 5.2. Section 5.3 discusses how SDD poses the problem of decoding. Magnitude $m$ for each bit of the symbol (discarded by HDD) is used as a measure of reliability of the bit value in SDD. Exploiting SDD in RS coded storage centers on optimal identification of data shares, based on the reliability information from available servers, for reconstructing the original data in retrievals. Section 5.4 describes adapting SDD to RS coded storage for STI based server reliability. To the best of our knowledge quantifying server
reliability / trust as ‘server trust index’ and exploiting SDD based on server reliability (trust) is the first attempt of this type in DSS-D research and is a significant contribution of the thesis.

5.2 THE DECODING PROBLEM

![Figure 5.1 Model of a coded system [40]](image)

A simplified model of coded system [40] is illustrated in Figure 5.1. A digital source is interested in communicating information (text, Video, audio) to the digital sink via a channel. With digital communication, bits of information are combined into symbols – symbol sequences form the baseband signal. The information sequence, \( u \) is a sequence of binary digits (bits) of the information which is given to the encoder. The symbol sequence is encoded using an error control coding scheme. The encoder processes \( u \) into a discrete encoded sequence called the codeword \( v \). There is a one-to-one correspondence between \( u \) and \( v \). The encoded symbol sequence is modulated on the carrier and transmitted. The noisy channel may corrupt (disturbances, dust, defects) the modulated carrier. At the receiver, the demodulator processes the received carrier to produce the sequence of demodulated outputs \( r \) representing the transmitted \( v \). The decoder receives the possibly corrupt binary sequence \( r \)
from the demodulator and performs decoding. The decoding problem is that an estimate \( \hat{u} = u \) must be produced based on \( r \). Decoding in effect must generate an estimate \( \hat{\nu} = \nu \) for each possible received sequence \( r \). Obviously \( \hat{u} = u \) iff \( \hat{\nu} = \nu \) and decoding error occurs iff \( \hat{\nu} \neq \nu \).

5.2.1 Hard Decision Decoding (HDD)

Decoding is primarily ‘hard’. Decoding basically takes a stream of bits say from the 'threshold detector' stage of a receiver, where each bit is considered definitely one or zero.

An HDD poses the problem of decoding as solely:

\[
\text{Given the set / vector } \{r_{n-1}, r_{n-2}, ..., r_1, r_0\}, \\
\text{identify the most likely / best possible code word}.
\]

The (possibly) corrupted modulated carrier received by the receiver is demodulated to generate a symbol sequence representing the encoded symbol sequence at the transmitter. In fact the demodulator outputs a number for each bit of the symbol. The sign of the number is mapped as the corresponding bit. With HDD, the number magnitude \( m \) is ignored. This – ‘hard bits’ – is fundamental to the use of HDD algorithms such as the Berlekamp-Massey and the Euclidian algorithms. Restricting ourselves to the block codes – \((n, k)\) code – the information symbol sequence is split and grouped into a sequence of \( k \) information symbols each. Each such \( k \)-symbol set is encoded into a corresponding \( n \)-symbol set. For convenience in analysis each encoded \( n \)-symbol set \( \{c_{n-1}, c_{n-2}, ..., c_1, c_0\} \) is represented as a corresponding polynomial – \( c(x) \) – of degree \( n - 1 \) as 
\[
c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ... + c_1x + c_0
\]
\( c(x) \) is called the ‘transmitted polynomial’. This constitutes the introduction of measured and uniform redundancy into the information sequence. The demodulated bit sequence is suitably split into a sequence of bit sets, each bit-set identified as a symbol, and
the received symbol sequence formed. Each such received symbol set is represented as a polynomial \( r(x) \) with \( \{ r_{n-1}, r_{n-2}, ..., r_1, r_0 \} \) being the received symbol set. For an \((n, k)\) RS code, the code set \( \{ C(x) \} \) is the collection formed with all possible \( k(x) \). With the coding schemes in general use, every pair of elements – \( c_i(x), c_j(x) \) – in the code set differ in at least \( d_{\text{min}} \) coefficients (\( d_{\text{min}} \) of RS code is \( n - k + 1 \)). Given \( r(x) \) with coefficients \( r_i \), decoding is the process of identifying an appropriate \( e(x) \) such that \( e(x) + r(x) \in \{ C(x) \} \).

### 5.3 SOFT DECISION DECODING (SDD)

HDD essentially ignores the associated magnitude \( m \) with each bit of the demodulated output. It should be noted that the magnitude \( m \) discarded by HDD, is in fact a measure of the confidence level of the decision in assigning the bit value. With a channel where the corrupting noise is treated as additive Gaussian, the probability of the decision in the bit value assignment is the shaded area of \( A \) in the Figure 5.2.

![Figure 5.2 Distribution of bit magnitude \( m \)](image)

In contrast to HDD, Soft Decision Decoding requires a stream of 'soft bits' where we get not only the 1 or 0 decision but also an indication of how certain we are that the decisions are correct. It should be noted that the decisions are likely to be much better with the greater
reliability being placed on bits we are certain about than those we are not so certain about. As shown in Figure 5.2, \( p(x) \) – the probability of the bit value being correct represented by the area \( A \) – can be directly used as the reliability value itself. In turn \( m \) itself can be used as the measure of the reliability of the bit value [40]. The fact that \( c(x), r(x), \) and \( e(x) \) of RS codes have coefficients in \( GF(q) \) and not in \( GF(2) \) entail the use of symbol reliability for \( e(x) \) identification. Associated bit reliability can be suitably combined to yield the symbol reliability. Thus each \( r_i \) in a set \( \{r_{n-1}, r_{n-2}, \ldots, r_1, r_0\} \) can have associated symbol reliability. An SDD poses the problem of decoding as:

‘Given the set / vector \( \{r_{n-1}, r_{n-2}, \ldots, r_1, r_0\} \) and an associated reliability set / vector \( \{s_{n-1}, s_{n-2}, \ldots, s_1, s_0\} \), identify the most likely / best possible code word’.

Different approaches to SDD based on reliability are feasible. SDDs differ the way reliability information is used along with the generator polynomial / matrix to extract a \( c(x) \). SDDs identify and process the most reliable or least reliable zones (Identifying Least Reliable Independent Positions (LRIPs) and Most Reliable Independent Positions (MRIPs) as per the decoding schemes. LRIPs and MRIPs are identified by reordering the received symbols according to their reliability.

A few representative approaches take one of the following routes (a, b, c) after the received vector is rearranged in order of reliability. Short listing of possible candidate code words often forms an intermediate step.

a. Take out a subset from the rearranged set; start from the most reliable end and extract a \( c(x) \) from this.

b. From the rearranged set above discard the symbols from the least reliable end successively or in combinations and identify a \( c(x) \) from the rest.
c. Shuffle the received vector and the generator matrix in specific combinations and combine with corresponding algorithms to arrive at a $c(x)$.

Generalized Minimum Distance decoding algorithm and Chase algorithms process LRIPs of the reordered sequence to search for the codewords. [17] processes MRIPs. List decoding algorithms [40] is of category (c) above. To give flair of SDD, the steps involved in Chase 3 algorithm for the binary case are given below:

1. Form $r(x)$.
2. If $d_{min}$ is even modify $r(x)$ by complementing no bit, the least reliable bit, the least three reliable bits, . . . , and the least $d_{min} - 1$ reliable bits. If $d_{min}$ is odd do the same with the least two reliable bits, the least four reliable bits, . . and the $d_{min} - 1$ least reliable bits.
3. Decode each of the above sets using HDD.
4. From the decoded set in (3) above identify the most likely codeword.

Chase 2 and Chase 1 algorithms are more involved and remain essentially of academic interest. In general the SDDs which do decoding starting at the most reliable end are all adaptations / simplifications of the Chase 3 algorithm.

In essence, SDD algorithms generally aim at generation of a set of codewords with the most likely codeword being present in it with high probability; then the same is selected based on a suitable likelihood measure. The complexity and performance of the techniques are influenced by the number of candidate codewords in the set. SDD algorithms for RS codes (Chase and GMD extensions for non-binary codes; Chase-GMD [60], ML-SDD [62, 49] and List decoding algorithms [24, 58, 70]) enhance the performance further, trading off complexity [40]. In general, the algorithms trade off complexity for computing effort. Conventional hard decoding is restricted to cases where the Hamming weight of $e(x)$ is
\[ d_{\text{min}}/2 \]. In contrast to HDD, SDD can extend the decoding range beyond the \([d_{\text{min}}/2] \) range with reliability as the basis and identify the most probable codeword starting with the received word. Appendix C, illustrates a list of the most reliable codewords nearest to the received word for an example of SDD of (7, 5) RS code.

5.4 SDD IN RS CODED STORAGE

The work presented here introduces ‘Server Trust Index’ (STI) as a metric for server reliability based on cumulative fragment share submission status in the past performances of storage servers. This rightly enables trust based selection of the most reliable (minimal) server set to provide ‘trust-based select fragments’ for file reconstruction (Vide Chapter 4). In contrast to HDD, SDD primarily requires a stream of ‘soft bits’ (reliability information) to perform decoding (Vide Section 5.3). Exploiting SDD in RS coded storage centers on optimal identification and processing of data shares based on the reliability information from available servers (for reconstructing the original data in retrievals).

5.4.1 Server and symbol reliability

We know that SDDs for RS codes make use of symbol reliability to identify reliable symbols in extracting the codeword; evidently, bit reliabilities are combined to yield the \( m \)-bit symbol reliability. Storage servers store and supply fragment shares for data retrievals and it is this (storage server) entity that acquires the – STI – reliability index in the ‘trust based RS coded storage’ design. Each fragment share stored at servers in the RS-coded storage here is a \( n \)-symbol share and in turn each symbol is a \( m \)-bit symbol. To exercise SDD at the RS-coded storage, explicit ‘reliability assignment’ is required for the \( n \)-symbol shares and equally to the \( m \)-bit symbols in the share. This necessitates propagation of server level reliability to the symbol level; Figure 5.3 illustrates this propagation from the storage entity (server) down to
the stored entity (n-symbol fragment share) and further down to the contained entity (m-bit symbol).

![Diagram of reliability propagation](image)

**Figure 5.3 Server reliability propagation**

As exemplified in Figure 5.3; Every $l^{th}$ server – that stores the $j^{th}$ fragment share for the $i^{th}$ file at storage – has reliability index $l_{STI}$. $l_{STI}$ is derived from the cumulative fragment share submission status in the past retrievals. A fragment share is in fact as reliable as the server that stores it; fragment share reliability at the $l^{th}$ server is equally $l_{STI}$. Server level reliability is first propagated to the share level. Symbols of an $n$-symbol share are in fact as reliable as the share itself; symbol reliability of an $n$-symbol share at the $l^{th}$ server is equally $l_{STI}$. Fragment share reliability is next propagated to the symbol level. Reliability can also be proliferated to the bit level of every $m$-bit symbol in an $n$-symbol fragment share; every bit is in fact as reliable as the server that stores it. Bit level propagation is redundant here in the RS
coded storage. Thus, reliability proliferates from the storage (server) entity down to the stored
(n-symbol fragment share) entity and further down to the contained (m-bit symbol) entity.

5.4.2 Data retrieval

Data retrievals in RS-coded storage in effect demand that k reliable servers be selected from
the set of (k + δ) malicious and non-malicious available servers, to make available the most
reliable fragment shares for reconstruction. Iterative decoding in file reconstructions
seamlessly integrate with the subjective approach to server selections. An iterative approach
to reconstruction at data retrievals is as follows:

Step 1: Server arrangement

Arrange available (k + δ) servers in the descending order of STI to form the ordered
list of servers.

Segregate the servers into three categories based on the STI distribution – most
reliable, moderately reliable, and the least reliable.

Step 2: Server selection

Select a set of first k reliable servers starting from the most reliable category and if
required moving to the other categories as well.

Step 3: Data reconstruction

Form an M × k matrix of fragment byte sets with the k fragment shares from the set
of k reliable servers in Step 2.

Repeat for all j from 1 to M to generate the reconstructed file

Form the k equations in the unknown symbol set {m₀,m₁, ...,m_{k-1}} and
solve.
Each row of $M \times k$ matrix – a collection of $k$ encoded symbols – is used with the coefficients of $g(x)$ to from the $k$ equations.

The solution is the set $\{b_{j,0}, b_{j,1}, ..., b_{j,k-1}\}$, the $j^{th}$ block $B_j$ of the original $F$ which was dispersed.

Perform file level integrity check – CRC checksum – (Vide Chapter 4) on the reconstructed file

If (file level integrity)

then ‘Successful reconstruction’

Return

else Perform Step 4.

*Step 4: Combination of server selection*

While ( ! file level integrity)

Identify the next server combination which is most likely to form the correct set of fragment byte sets from the server arrangement in Step 1.

Perform Step 3.

Reconstruction complexity in retrievals is directly related to identifying $k$ reliable servers to provide the $k$ fragment shares for successful reconstruction. There is always a probability that servers (with relatively high STI) classified as most reliable / moderately reliable for the current retrieval, may have been subjected to a compromise / error most recently between the preceding and the current retrieval. This invariably leads to server misclassification in server segregations, accounting for ‘false positives’ ($fp$) in the reliable categories. Number of attempts expended in finding the set of $k$ reliable servers for successful reconstruction is of the order of $(k ^ fp)$. While STI allows for effective segregation, these false positives are accountable for the repeated attempts at data reconstruction.
SDD algorithms provide similar iterative procedures for data reconstruction. Set of 
\((k + \delta)\) servers available for data retrieval provide \((k + \delta)\) fragment shares for data 
reconstruction. Reliability is duly proliferated from STI of the corresponding server down to 
the \(m\)-bit symbol level of the shares. A set of \(M\) blocks (of each \((k + \delta)\) encoded bytes) is 
generated from the \((k + \delta)\) fragment byte sets. \(M\) blocks of the file and the associated set of 
symbol reliability values form the input to SDD algorithm. For each block, the algorithm 
outputs the most probable set of \(n\) encoded bytes of the original. The file may be assembled 
from \(M\) such outputs; this constitutes the reconstructed file.

Short listing of possible candidates often forms an intermediate step in SDD algorithms. 
For each of the candidate set of \(M\) such outputs in the short listings, file may be reconstructed 
and verified for integrity check. A file level integrity check on the reconstructed file confirms 
it as the exact copy of the original file dispersed at storage. Failure at integrity check – if it 
happens with every possible \(k\)-combination of servers out of the \((k + \delta)\) set – means an 
overall file reconstruction failure at this step / stage. It asserts that among the available 
\((k + \delta)\) servers for data retrieval the number of erroneous servers is \(\geq k\) and any attempt(s) 
to reconstruct the file with these servers may be futile.

A representative approach to reconstruction at data retrievals using SDD is as follows:

\textit{Step 1:} Arrange available \((k + \delta)\) servers in the order of STI to form the ordered list 
of servers. Generate \(M\) blocks of the file and the vector of symbol reliability values 
from these servers.

\textit{Step 2:} Perform SDD with the \(M\) blocks and the symbol reliability vector\(^1\) from Step 1 
as input. Candidate sets for each \(M\) block is output.

\(^1\) Every set of received symbols has an associated set of reliability values (reliability vector) for SDD in 
communication. However, in the RS coded storage here, SDD receives a single reliability vector (generated 
from the participating \((k + \delta)\) servers) which remains invariant for all the \(M\) blocks of encoded bytes.
Step 3: Starting from the most likely end, for each of the candidate set of $M$ blocks output in Step 2, perform the following.

Reconstruct the file

Perform file level integrity check – CRC checksum – on the reconstructed file

If (file level integrity)

then ‘Successful reconstruction’; break;

Return

In digital communication every set / group of $k$ bytes encoded to $n$ bytes is a separate entity. At the receiver each of these $n$ bytes has its own reliability value which is used in SDD. The SDD is to be done for each such $n$-byte entity separately to recover the message. The adaptation presented so far follows a similar procedure for file reconstruction. But with DSS-D, each file share as a whole has one reliability value – the STI of the server which returned the file share. Thus every one of the bytes in the set \( \{N_{1,i-1}, N_{2,i-1}, \ldots, N_{M,i-1}\} \) returned by the $i^{th}$ server (Vide Section 3.4.2) has the same reliability value. This can be fruitfully exploited for data retrieval. The scheme is shown in Figure 5.4. It entails a minor modification of the scheme presented earlier. The steps in file write and read follow.

A. File write

1. Split the file into $M$ byte sets. To each byte set append its CRC bit sequence; the appended byte set is $k$ bytes long.

2. Encode the $M$ byte sets into respective code sets each of $n$ bytes.

3. Form the $n$ file shares – each of $M$ bytes as explained earlier – and distribute them to the $n$ servers at the storage tier.
Figure 5.4 Block level CRC for reconstruction
Doing CRC and appending it to the byte set as in (1) above imply a corresponding increase in computational effort as well as storage size; but this increase is marginal.

**B. File Read**

1. Use STI values and identify the most reliable $k$ set of servers as explained in Section 5.4.2. For brevity in depiction these are assumed to be the servers with indices 1 to $k$ in the Figure 5.4.

2. With the $k$ encoded bytes of the first $n$ encoded byte set reconstruct the first byte set; this is the retrieved byte set with its CRC bit sequence appended.

3. Verify the correctness of the set by computing the CRC afresh and checking it.
   In case of a CRC mismatch try the next $k$ set; repeat the procedure outlined as in Section 5.4.2 until the correct byte set is retrieved.

4. In case the byte set retrieval in (3) above is based on an identified $k$-set of servers (as is the case in the presented work) use these same $k$ servers as the basis and retrieve all subsequent byte sets.

5. With each of the byte sets retrieved the CRC computation and check may be repeated.
   In fact these are essentially checks of reconfirmation.

6. If the CRC check in (5) above fails, one has to fall back on SDD based data retrieval process as in Section 5.4.2 itself.
   But, this eventuality has so low a probability as to be considered rare.

The procedure here calls for the iterative reconstruction procedure only for the first byte set. For all the subsequent byte sets data retrieval is direct and involves only minimal computational effort.
5.5 CONCLUSION

In the presence of malicious activities at storage, ‘equality in trust’ is certainly not practical; further, decoding is primarily ‘hard’. There is no scope for trust / reliability information in HDD. Exercising HDD in DSS-D during data retrievals, compels equal treatment of storage servers and their fragment shares in reconstructions (decoding). In contrast to HDD, Soft Decision Decoding accommodates confidence levels in the bit / symbol set. Exercising SDD in DSS removes this unwarranted compulsion of treating storage servers and their fragment shares equally in reconstructions (decoding).

SDD for RS codes makes use of symbol reliability to identify reliable symbols in extracting the codeword. To exercise SDD at RS coded storage ‘reliability assignment’ is indispensable for the m-bit symbols of fragment shares stored at servers. An organized approach to compute and administer ‘server trust’ – a reliability measure – effectively across data retrievals is well primed (Vide Chapter 4). In the RS-coded storage here, reliability is suitably propagated from the server entity level down to the symbol level of fragment shares it stores, making it most appropriate for application of SDD.

It is essential to investigate and validate the efficacy of ‘trust based server and fragment share selections’ at data retrievals and consequently its effect in equally enabling a secure and reliable storage system. A well structured simulation scheme to observe and analyze Fault Tolerant-Storage Access Framework (FT-SAF) of representative distributed storage systems is needed\(^2\). This forms the scope of the next chapter.

\(^2\) Available simulation tools (P2PSim and the like) are not adequately suited for analysis of schemes with selective interactions in hybrid failures (crash and non-crash faults) at storage – the focus of the presented work.