IMPLEMENTASI KODE REED-SOLOMON UNTUK DETEKSI DAN KOREKSI KESALAHAN TRANSMISI AYAT AL-QUR'AN MENGGUNAKAN PENGKODEAN HURUF HIJAIYAH

SKRIPSI

OLEH
TAHIRA KHUWALIDIA SHABIRAH MA'RUF
NIM. 210601110087

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2025

IMPLEMENTASI KODE REED-SOLOMON UNTUK DETEKSI DAN KOREKSI KESALAHAN TRANSMISI AYAT AL-QUR'AN MENGGUNAKAN PENGKODEAN HURUF HIJAIYAH

SKRIPSI

Diajukan Kepada
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
untuk Memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Matematika (S.Mat)

Oleh
Tahira Khuwalidia Shabirah Ma'ruf
NIM. 210601110087

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2025

IMPLEMENTASI KODE REED-SOLOMON UNTUK DETEKSI DAN KOREKSI KESALAHAN TRANSMISI AYAT AL-QUR'AN MENGGUNAKAN PENGKODEAN HURUF HIJAIYAH

SKRIPSI

Oleh
Tahira Khuwalidia Shabirah Ma'ruf
NIM. 210601110087

Telah Disetujui Untuk Diuji

Malang, 10 Desember 2025

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Khudzaifah, M.Si.	Dr. Fachrur Rozi, M.Si.
NIPPPK. 19900511 202321 1 029	NIP. 19800527 200801 1 012

Mengetahui,
Ketua Program Studi Matematika

Dr. Fachrur Rozi, M.Si.
NIP. 19800527 200801 1 012

IMPLEMENTASI KODE REED-SOLOMON UNTUK DETEKSI DAN KOREKSI KESALAHAN TRANSMISI AYAT AL-QUR'AN MENGGUNAKAN PENGKODEAN HURUF HIJAIYAH

SKRIPSI

Oleh
Tahira Khuwalidia Shabirah Ma'ruf
NIM. 210601110087

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima sebagai Salah satu Persyaratan
untuk Memperoleh Gelar Sarjana Matematika (S.Mat)

Tanggal 22 Desember 2025

Ketua Penguji	: Prof. Dr. H. Turmudi, M.Si., Ph.D.	.....
Anggota Penguji 1	: Hisyam Fahmi, M.Kom.	.....
Anggota Penguji 2	: Muhammad Khudzaifah, M.Si.	.....
Anggota Penguji 3	: Dr. Fachrur Rozi, M.Si.	.....

Mengetahui,
Ketua Program Studi Matematika

Dr. Fachrur Rozi, M.Si.
NIP. 19800527 200801 1 012

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	:	Tahira Khuwalidia Shabirah Ma'ruf
NIM	:	210601110087
Program Studi	:	Matematika
Fakultas	:	Sains dan Teknologi
Judul Skripsi	:	Implementasi Kode Reed-Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya sendiri, bukan merupakan pengambilan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan dan pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila di kemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 22 Desember 2025
Yang membuat pernyataan,

Tahira Khuwalidia S.M
NIM. 210601110087

MOTO

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya”
-Q.S Al-Baqarah: 286

“Aku lahir dengan mempertaruhkan nyawa seorang Ibu, maka aku harus menjadi alasan semua pengorbanannya bermakna. Ayah lelah setiap hari demi langkahku, maka biarlah setiap tetes peluhnya berbuah bangga.”

“Skripsi ini mungkin tidak sempurna, namun ia adalah bukti perjalanan panjang yang mengantar langkahku gelar S.Mat. *Bismillah* untuk segala hal-hal baik yang sedang diperjuangkan.”

PERSEMBAHAN

Dengan penuh rasa syukur ke hadirat Allah SWT atas
segala rahmat, pertolongan dan kemudahan-Nya, sehingga skripsi ini dapat terselesaikan dengan baik. Karya ini penulis persembahkan kepada:

Ayah, Mama, dan Bunda, yang kasih sayangnya tak pernah berkurang, yang doanya menjadi kekuatan terbesar dalam setiap langkah, dan yang pengorbanannya takkan pernah mampu penulis balas dengan apa pun di dunia ini.

Saudara-saudari tercinta, yang senantiasa memberi doa, dukungan, dan semangat tanpa henti.

Orang terdekat penulis, yang hadir sebagai tempat berbagi cerita, keluh, dan harapan, serta menjadi sumber kekuatan saat penulis hampir menyerah.

Sahabat-sahabat terbaik, yang dengan tulus menemani, membantu, dan menyemangati hingga skripsi ini dapat diselesaikan. Terima kasih atas keikhlasan, kebersamaan, dan tawa yang membuat perjalanan ini lebih ringan.

Semoga setiap doa dan kebaikan kalian menjadi amal yang terus mengalir, dan semoga karya sederhana ini dapat menjadi kebanggaan bagi kita semua.

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulillahirabbilalamin, segala puji dan syukur senantiasa penulis panjatkan ke hadirat Allah subhanahu wa ta'ala atas berkat Rahmat, serta hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Implementasi Kode Reed-Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah" dengan baik dan benar. Shalawat serta salam tetap tercurahkan kepada Nabi Muhammad SAW, yang telah membawa kita dari zaman kebodohan menuju zaman kebenaran yakni Islam dan zaman yang penuh dengan ilmu pengetahuan sebagaimana yang dirasakan pada saat ini. Dan semoga kita semua mendapat syafaatnya di hari akhir kelak, Aamiin.

Penulis mengucapkan rasa terima kasih yang begitu besar kepada seluruh pihak yang memberikan dukungan dan motivasi kepada penulis sehingga dapat menyelesaikan skripsi ini. Ucapan terima kasih ini penulis sampaikan kepada:

1. Prof. Dr. Hj. Ilfi Nur Diana, M.Si., CAHRM, CRMP., selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang
2. Dr. H. Agus Mulyono, M.Kes., selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
3. Dr. Fachrur Rozi, M.Si., selaku Ketua Program Studi Matematika dan selaku Dosen Pembimbing II, Universitas Islam Negeri Maulana Malik Ibrahim. Terima kasih atas kesabaran, arahan, serta masukan yang berharga dalam proses penyelesaian skripsi ini. Semoga segala bimbingan yang diberikan menjadi pahala dan berkah bagi Bapak.
4. Muhammad Khudzaifah, M.Si., selaku Dosen Pembimbing I atas bimbingan, dukungan, dan arahan yang telah diberikan selama proses penyusunan skripsi ini. Semoga ilmu yang Bapak bagikan menjadi amal jariyah dan senantiasa bermanfaat.
5. Prof. Dr. H. Turmudi, M.Si., Ph.D., selaku Ketua Penguji, yang telah memberikan arahan, masukan, serta saran yang membangun demi penyempurnaan skripsi ini.

6. Hisyam Fahmi, M.Kom., selaku Anggota Penguji I dalam Ujian Skripsi, yang telah memberikan kritik, saran, dan masukan yang konstruktif guna perbaikan dan penyempurnaan skripsi ini.
7. Seluruh dosen Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim
8. Kepada Ayah M. Munif, sosok panutan sepanjang hidup penulis, atas kasih sayang, doa, nasihat, dan dukungan yang senantiasa diberikan selama penulis menempuh studi. Pengorbanan dan keyakinan beliau menjadi sumber semangat utama bagi penulis. Kepada Mama Lailatul Firdausi Ma'ruf, pintu surga penulis, atas ketulusan, kesabaran, serta doa dan nasihat yang senantiasa mengiringi setiap langkah penulis. Kepada Eyang Uti Maryam, yang dengan penuh kasih telah membimbing, merawat, dan menanamkan nilai-nilai kebaikan serta kedisiplinan ibadah sejak kecil. Kepada Bunda Yuliana, atas cinta, doa, dan ketulusan yang tetap mengalir meskipun terpisah oleh jarak. Semoga Allah SWT membalas seluruh kebaikan dan pengorbanan beliau dengan pahala dan keberkahan yang berlimpah.
9. Kepada adik-adik penulis, Chlorina Reva Nadia Ma'ruf, Muhammad Nabil Firdausi Ma'ruf, dan Alfa Rizqi Anggawa Ma'ruf, yang senantiasa menjadi sumber semangat, penguat hati, dan penyejuk di tengah perjuangan penulis menempuh studi. Semoga tumbuh menjadi pribadi yang kuat, cerdas, dan senantiasa dalam lindungan Allah SWT. Aamiin.
10. Kepada keluarga besar H. Adjik dan H. Mashuri yang telah menjadi bagian penting dalam perjalanan hidup penulis. Terima kasih atas perhatian, do'a, dan dukungan yang selalu diberikan, baik secara langsung maupun tidak langsung. Semoga silaturahmi dan kasih sayang di antara kita selalu terjaga, dan Allah senantiasa melimpahkan keberkahan untuk keluarga besar ini.
11. Kepada Muhammad Putra Zidannizar, sosok lelaki kedua setelah Ayah penulis yang senantiasa hadir sebagai pendamping dan penyemangat sejak awal perjalanan perkuliahan hingga saat ini. Terima kasih atas ketulusan, kesabaran, serta dukungan yang tak pernah putus, bahkan dalam masa-masa sulit sekalipun. Bersama, kami telah melewati berbagai suka dan duka, saling menguatkan, saling memahami, dan saling membantu satu sama lain dalam

menapaki dunia akademik maupun kehidupan sehari-hari. Semoga kebersamaan ini senantiasa terjaga, dan kelak menjadi jalan menuju masa depan yang diridhoi, penuh keberkahan, dan menjadi tujuan hidup bersama. Terima kasih telah menjadi bagian penting dari setiap langkah penulis.

12. Kepada sahabat-sahabat penulis, Elza Zanuarika Kusniadi, Fatma Nabila Nur Aini, Amalia Fitriani, Cetrin Aprilia, Dinda Nur Azizah, Putri Wahyu, serta seluruh anggota grup "OTDR", terima kasih atas kebersamaan, dukungan, dan semangat yang telah diberikan sejak masa sekolah hingga proses penyusunan skripsi ini. Kebersamaan, motivasi, dan saling menguatkan yang terjalin menjadi bagian berharga dalam perjalanan akademik penulis. Semoga persahabatan dan tali silaturahmi ini senantiasa terjaga serta membawa kebaikan dan keberkahan bagi kita semua.
13. Seluruh mahasiswa angkatan 2021 dan teman teman penulis atas kebersamaan, dukungan, serta motivasi yang telah diberikan selama masa studi ini. Kehadiran dan semangat kalian menjadi bagian penting dalam perjalanan ini. Semoga kebersamaan ini terus terjalin dan membawa manfaat bagi kita semua.
14. Untuk diri sendiri, Tahira Khuwalidia Shabirah Ma'ruf, yang telah melalui banyak proses jatuh, bangkit, lelah, namun tetap berjalan. Terima kasih telah bertahan sejauh ini, terus belajar, dan tidak menyerah meski kadang ingin berhenti. Semoga tetap konsisten dalam hal baik, terus berproses, dan tidak lupa tujuan awal. Ingat, segala sesuatu butuh waktu, dan selama terus melangkah, tidak ada usaha yang sia-sia. Percayalah, kerja keras dan doa tidak pernah mengkhianati hasil. Tidak ada kata terlambat untuk menciptakan kehidupan yang kamu inginkan.

Malang, 22 Desember 2025

Tahira Khuwalidia S.M

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR SIMBOL

C	:	Codeword
G	:	Matriks generator
H	:	Matriks *parity-check*
M	:	Blok pesan
t	:	Jumlah maksimum kesalahan bit yang dapat dikoreksi
n	:	Panjang total bit
k	:	Panjang pesan asli
m	:	Jumlah bit dalam satu simbol

DAFTAR LAMPIRAN

ABSTRAK

Ma'ruf, Tahira Khuwalidia Shabirah. 2025. **Implementasi Kode Reed–Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah.** Skripsi. Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Muhammad Khudzaifah, M.Si. (II) Dr. Fachrur Rozi, M.Si.

Kata Kunci: Reed–Solomon, Huruf Hijaiyah, Deteksi Kesalahan, Koreksi Kesalahan, Transmisi Digital.

Kesalahan penulisan huruf hijaiyah dalam ayat Al-Qur'an berpotensi mengubah makna ayat serta menurunkan keakuratan penyampaian teks. Penelitian ini bertujuan untuk mendeskripsikan proses deteksi dan koreksi kesalahan transmisi ayat Al-Qur'an menggunakan kode Reed–Solomon. Ayat Al-Qur'an direpresentasikan dalam bentuk kode Unicode huruf hijaiyah sebagai data digital. Data penelitian terdiri atas penulisan sepuluh ayat Al-Qur'an yang digunakan sebagai data uji, dengan penambahan kesalahan secara sengaja dan terkontrol pada tahap transmisi untuk mensimulasikan kondisi transmisi digital. Setiap huruf hijaiyah dikonversi kedalam representasi *Unicode* 16-bit, kemudian dilakukan proses *encoding* menggunakan parameter kode $RS(n, k, 2t)$. Selanjutnya, kesalahan disisipkan secara terkontrol untuk mensimulasikan gangguan transmisi, dan proses *decoding* dilakukan menggunakan algoritma Reed–Solomon berbasis *Galois Field* melalui perangkat lunak SageMath. Hasil analisis menunjukkan bahwa kode Reed–Solomon mampu mendeteksi dan mengoreksi kesalahan simbol sesuai dengan kapasitas koreksi t secara konsisten pada berbagai variasi jumlah *error*. Pada seluruh skenario pengujian, huruf hijaiyah yang mengalami gangguan berhasil dikembalikan ke bentuk aslinya melalui tahapan perhitungan sindrom, pembentukan polinomial lokasi kesalahan, perhitungan polinomial evaluator, hingga proses koreksi akhir pada *codeword*. Penelitian ini membuktikan bahwa kode Reed–Solomon dapat berfungsi sebagai mekanisme verifikasi yang efektif untuk menjaga keakuratan teks Al-Qur'an dalam sistem digital serta mendukung upaya pelestarian kemurnian ayat melalui pendekatan teori pengkodean.

ABSTRACT

Ma'ruf, Tahira Khuwalidia Shabirah. 2025. **Implementation of Reed–Solomon Codes for Error Detection and Correction in the Transmission of Qur'anic Verses Using Hijaiyah Character Coding.** Thesis. Department of Mathematics, Faculty of Science and Technology, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Advisors: (I) Muhammad Khudzaifah, M.Si. (II) Dr. Fachrur Rozi, M.Si.

Keywords: Reed–Solomon, Hijaiyah Characters, Error Detection, Error Correction, Digital Transmission.

Errors in the representation of Hijaiyah characters in Qur'anic verses may alter the semantic meaning of the text and reduce transmission accuracy. This study aims to describe the error detection and error correction processes in the digital transmission of Qur'anic verses using Reed–Solomon codes. The Qur'anic verses are modeled as digital data by encoding Hijaiyah characters into their corresponding Unicode representations. The research data consist of ten Qur'anic verses used as test data, in which errors are intentionally and controllably introduced during the transmission stage to simulate digital transmission conditions. Each Hijaiyah letter is converted into a 16-bit Unicode representation, followed by an encoding process using Reed–Solomon code parameters $RS(n, k, 2t)$. Subsequently, controlled errors are inserted to model transmission disturbances, and the decoding process is performed using a Reed–Solomon algorithm over a Galois Field implemented in SageMath software. The results show that the Reed–Solomon code is capable of consistently detecting and correcting symbol errors in accordance with its error-correction capability t under various error scenarios. In all test cases, corrupted Hijaiyah characters are successfully restored to their original form through syndrome computation, construction of the error locator polynomial, evaluation of the error evaluator polynomial, and the final correction of the received codeword. This study demonstrates that Reed–Solomon codes can serve as an effective verification mechanism to preserve the accuracy of Qur'anic text in digital systems and support efforts to maintain the integrity of Qur'anic verses through coding theory approaches.

مستخلصا البحث

معروف، تاهيرا خووإليديا شابيره. ٢٠٢٥ .تطبيق شفرة ريد–سولومون لاكتشاف وتصحيح الأخطاء في نقل آيات القرآن الكريم باستخدام ترميز الحروف الهجائية العربية .البحث الجامعي. قسم الرياضيات، كلية العلوم والتكنولوجيا، جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: (١) محمد خذيفة، الماجستير في العلوم. (٢) الدكتور. فخر الرّازي، الماجستير في العلوم.

الكلمات الأساسية :ريد–سولومون، الحروف الهجائية العربية؛، اكتشاف الأخطاء، تصحيح الأخطاء، النقل الرقمي.

إنَّ الأخطاء في كتابة الحروف الهجائية في آيات القرآن الكريم قد تؤدي إلى تغيّر المعنى وتقليل دقّة نقل النص. وتهدف هذه الدراسة إلى وصف عملية الكشف عن الأخطاء وتصحيحها في نقل آيات القرآن الكريم باستخدام شفرة (Reed–Solomon) ، وتمثَّل آيات القرآن الكريم في هذه الدراسة في صورة بيانات رقمية بالاعتماد على ترميز (*Unicode*) للحروف الهجائية، حيث تتكوّن بيانات البحث من كتابة عشر آيات من القرآن الكريم استُخدمت كبيانات اختبار، مع إدخال أخطاء بشكل متعمّد ومُتحكَّم فيه في مرحلة الإرسال، وذلك لمحاكاة ظروف الإرسال الرقمي، ويُحوَّل كل حرف هجائي إلى تمثيل يونيكود بطول 16 بت، ثم تُجرى عملية الترميز باستخدام معاملات شفرة ريد–سولومون من النوع $RS(n, k, 2t)$، وبعد ذلك تُدرج الأخطاء بشكل مُتحكَّم فيه لمحاكاة اضطرابات الإرسال، وتُنفَّذ عملية فك الترميز باستخدام خوارزميات ريد–سولومون المعتمدة على الحقول المنتهية (*Galois Field*) من خلال برنامج SageMath ، وظهرت نتائج التحليل أن شفرة ريد–سولومون قادرة على الكشف عن أخطاء الرموز وتصحيحها بما يتوافق مع سعة التصحيح t وبصورة متّسقة عبر مختلف سيناريوهات عدد الأخطاء، حيث أُعيدت في جميع حالات الاختبار الحروف الهجائية التي تعرّضت للاضطراب إلى صورتها الأصلية من خلال مراحل حساب المتلازمات، وبناء كثير حدود تحديد مواقع الأخطاء، وحساب كثير حدود مُقيِّم الأخطاء، وصولًا إلى عملية التصحيح النهائية للكلمة المشفَّرة، مما يثبت أن شفرة ريد–سولومون يمكن أن تعمل كآلية تحقق فعّالة للحفاظ على دقة نصوص القرآن الكريم في الأنظمة الرقمية، وتسهم في دعم جهود صون سلامة الآيات والمحافظة على نقائها من خلال منهجية نظرية الترميز.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Ketepatan dalam penulisan huruf hijaiyah merupakan aspek yang sangat penting, khususnya dalam penulisan ayat Al-Qur'an, karena kesalahan dalam penulisan dapat mengubah bentuk lafaz dan berpotensi memengaruhi makna ayat. Dalam kajian penulisan teks suci Islam, bentuk tertulis dari ayat-ayat Al-Qur'an dikenal dengan istilah mushaf Al-Qur'an, yaitu kumpulan ayat Al-Qur'an yang dituliskan dan dibukukan secara sistematis sebagai representasi dari bacaan yang telah terjaga keasliannya. (Shihab, 2007).

Pada masa Rasulullah SAW, wahyu yang diturunkan belum dibukukan dalam satu mushaf yang utuh karena proses pewahyuan masih berlangsung. Oleh sebab itu, penjagaan Al-Qur'an dilakukan melalui hafalan para sahabat serta pencatatan ayat-ayat pada berbagai media sederhana. Setelah Rasulullah SAW wafat pada tahun 632 M, upaya penghimpunan ayat-ayat Al-Qur'an dilakukan pada masa Khalifah Abu Bakar *Ash-Shiddiq* dan kemudian disempurnakan pada masa Khalifah Utsman bin Affan melalui standarisasi mushaf yang dikenal sebagai Rasm Utsmani.

Seiring dengan perkembangan sejarah penulisan mushaf, dilakukan berbagai penyempurnaan teknis seperti penambahan tanda titik, harakat, dan tanda baca. Penyempurnaan tersebut bertujuan untuk menjaga kejelasan dan ketepatan penulisan ayat serta menghindari kesalahan pembacaan, tanpa mengubah substansi ayat. Dalam perkembangan selanjutnya, istilah Al-Qur'an digunakan secara luas

oleh masyarakat untuk merujuk tidak hanya pada wahyu dalam bentuk bacaan, tetapi juga pada mushaf sebagai bentuk tertulisnya.

Perkembangan dalam aspek penulisan Al-Qur'an tersebut sejalan dengan janji Allah SWT dalam Surah Al-Hijr ayat 9 (Kementerian Agama, 2022):

اِنَّ نَحْنُ نَزَّلْنَا الذِّكْرَوَاِنَّالَهُ لَحَافِظُوْنَ.

"Sesungguhnya Kami yang menurunkan Al-Qur'an, dan sesungguhnya Kami benar-benar menjaganya."(Qs. Al-Hijr : 9).

Ayat ini menegaskan bahwa penjagaan Allah SWT terhadap Al-Qur'an tidak hanya bersifat spiritual, tetapi juga berlangsung secara historis dan teknis, termasuk dalam aspek penulisan mushaf dan transmisi teks. Dalam perkembangan teknologi digital saat ini, tantangan penjagaan tersebut juga mencakup proses penulisan, penyimpanan, dan distribusi teks ayat Al-Qur'an dalam bentuk digital. Kesalahan penulisan huruf hijaiyah dalam sistem digital berpotensi mengubah makna ayat dan menurunkan keakuratan penyampaian teks.

Di era modern, upaya menjaga kemurnian Al-Qur'an terus dilakukan, baik melalui publikasi mushaf standar oleh lembaga resmi seperti Lajnah Pentashihan Mushaf Al-Qur'an (LPMQ), pelatihan dan sertifikasi qari dan hafidz, maupun pengembangan aplikasi digital Al-Qur'an yang dilengkapi fitur tajwid. Namun demikian, kesalahan penulisan ayat Al-Qur'an tetap dapat terjadi. Misalnya, dalam proses pengiriman atau pertukaran data ayat Al-Qur'an melalui media digital, seperti aplikasi pesan, media sosial, atau sistem penyimpanan, teks ayat dapat mengalami gangguan (*noise*) yang menyebabkan huruf berubah, hilang, atau tertukar. Kesalahan ini dapat muncul akibat konversi format teks, kerusakan file, atau ketidakcocokan sistem *encoding* karakter Arab. Meskipun tampak sederhana,

perubahan satu huruf saja dapat memengaruhi makna dan keutuhan teks ayat. Kesalahan serupa juga ditemukan pada media cetak. Salah satu contohnya adalah kasus kesalahan cetak dalam mushaf Al-Qur'an pada surah Al-Kahfi ayat 8, di mana huruf *'ain* (ع) diganti dengan *ha* (ه), sehingga kata *lajaa'iluuna* tertulis sebagai *lajaahiluuna* (Kemenag, 2023). Selain itu beberapa buku Pendidikan Agama Islam juga dilaporkan mengandung kesalahan penulisan ayat Al-Qur'an, sebagaimana diberitakan oleh Republika pada tanggal 29 Oktober 2017 (Republika, 2017). Fenomena ini menunjukkan bahwa keakuratan dalam penulisan Al-Qur'an tetap menjadi tantangan, meskipun teknologi telah berkembang pesat.

Untuk meminimalisir kesalahan serupa di era digital, diperlukan representasi huruf Al-Qur'an dalam bentuk numerik agar dapat diolah oleh sistem komputer. Dalam hal ini, huruf hijaiyah digunakan sebagai simbol yang direpresentasikan dalam bentuk kode numerik. Setiap huruf hijaiyah memiliki nilai *Unicode* yang berbeda, yang kemudian dapat dikonversi ke dalam bilangan desimal maupun biner. Dengan representasi ini, huruf-huruf hijaiyah dalam Al-Qur'an dapat digunakan sebagai data digital yang siap diproses menggunakan kode Reed-Solomon. Sebagai contoh, huruf hijaiyah ب (ba) direpresentasikan dalam *Unicode* dengan kode *U+0628* dengan nilai desimal 1576. Representasi ini selanjutnya dikonversi ke dalam biner 16-bit, yaitu 0000011000101000, sehingga dapat digunakan dalam proses *encoding* dan *decoding*. Melalui mekanisme ini, setiap huruf Al-Qur'an dapat diubah ke dalam bentuk biner standar yang konsisten, sehingga lebih mudah dikelola dalam proses deteksi dan koreksi kesalahan.

Salah satu pendekatan yang relevan untuk mengatasi permasalahan pada kasus di atas adalah dengan menerapkan teori pengkodean (*coding theory*). Teori

ini mempelajari karakteristik dan aplikasi kode dalam berbagai sistem, seperti kompresi data, kriptografi, dan kode pengoreksi *error* (*error-correction codes*). Tujuan utama dari teori pengkodean adalah untuk memberikan kode dengan tingkat informasi yang tinggi, kemampuan koreksi kesalahan yang kuat dan kompleksitas *encoding* dan *decoding* yang rendah (Widiastuti dkk., 2016). Dalam sistem komunikasi digital, proses pengkodean terbagi menjadi dua proses. Proses pertama adalah *encoding* (mengubah pesan menjadi *codeword)* dan proses kedua adalah *decoding* (mengembalikan *codeword* menjadi pesan asli), serta mengantisipasi kemungkinan kesalahan selama transmisi data (Oktavia dkk., 2023). Salah satu gangguan utama kesalahan dalam transmisi data adalah *noise*. *Noise* (derau) merupakan suatu sinyal pengganggu atau perusak sinyal, sehingga perlu dilakukan penghilangan *noise* agar sinyal informasi akan terpisah dari *noise*.

Dalam proses pengiriman sinyal informasi ke penerima akan melewati suatu media transmisi. Transmisi merupakan pengiriman sinyal dalam sistem komunikasi digital. Dalam ilmu komunikasi data, data berarti informasi yang disajikan dalam bentuk isyarat digital biner. Transmisi data merupakan proses pengiriman informasi di antara dua titik menggunakan kode biner melewati saluran transmisi dan peralatan *switching*, bisa antara komputer dan komputer, komputer dengan terminal, atau komputer dengan perlatan, atau peralatan dengan peralatan. Pada proses pengiriman ini maka akan muncul *noise* sehingga mengakibatkan sinyal informasi yang diterima mengalami gangguan dan bercampur dengan sinyal-sinyal yang tidak diinginkan sehingga dapat mengganggu keaslian informasi (Darmadi dkk., 2020).

Dalam penelitian ini, istilah transmisi tidak hanya dimaknai sebagai pengiriman sinyal dalam sistem komunikasi digital, tetapi juga sebagai proses penyampaian teks ayat Al-Qur'an dari satu media ke media lain, seperti dari mushaf cetak ke aplikasi digital atau dari hasil pengetikan manual ke database komputer. Pada proses ini, kemungkinan terjadi kesalahan pengiriman penulisan ayat Al-Qur'an dalam huruf hijaiyah dapat dipandang sebagai bentuk kesalahan (*error*) dalam transmisi data. Oleh karena itu, diperlukan mekanisme deteksi dan koreksi untuk memastikan teks yang diterima tetap sesuai dengan naskah aslinya. Deteksi kesalahan bertujuan untuk menemukan adanya perubahan atau kerusakan data selama proses transmisi, sedangkan koreksi kesalahan berfungsi untuk memperbaiki data yang rusak agar sesuai dengan aslinya. Dalam konteks transmisi ayat Al-Qur'an, deteksi dimaknai sebagai proses mengidentifikasi huruf hijaiyah yang mengalami perubahan atau ketidaksesuaian selama pengiriman. Adapun koreksi merujuk pada pengembalian huruf tersebut ke bentuk yang benar sesuai teks asli.

Salah satu metode yang banyak digunakan dalam deteksi dan koreksi kesalahan pada sistem komunikasi digital adalah kode Reed- Solomon. Kode Reed-Solomon diperkenalkan oleh Irving S. Reed dan Gustave Solomon pada tahun 1960. Reed-Solomon merupakan kode blok $RS(n, k)$ yang mampu mendeteksi dan mengoreksi kesalahan hingga sejumlah $t < (n - k)/2$ simbol. Kode ini banyak diterapkan pada sistem komunikasi satelit, pemutar CD/DVD, *QR Code*, serta sistem penyimpanan RAID karena efektivitasnya dalam menangani kesalahan dalam jumlah besar (Jariyah dkk., 2013). Selain itu, kode ini juga dikenal memiliki algoritma *encoding* dan *decoding* yang efisien (Oktavia dkk., 2023). Wicker dan

Bhargava menjelaskan bahwa terdapat tiga metode dalam membangun kode Reed-Solomon, yaitu menggunakan aritmatika pada lapangan hingga, polinomial generator, serta transformasi *Fourier* (Wicker, 2005).

Penelitian sebelumnya telah mengkaji berbagai metode untuk meningkatkan keamanan komunikasi dalam bahasa Arab dan mendeteksi kesalahan penulisan ayat Al-Qur'an. Alqahtani (2013), mengusulkan modifikasi algoritma *Vigenère cipher* menggunakan modulus 39 untuk enkripsi teks Arab, yang terbukti lebih aman dan efisien dibandingkan metode klasik. Riyanto (2019), menerapkan kode linear, khususnya kode Hamming berorde 3, untuk mendeteksi dan mengoreksi kesalahan satu huruf dalam penulisan huruf hijaiyah menggunakan representasi biner 5 *bit*, sehingga menjaga akurasi teks Al-Qur'an. Sementara itu, Oktavia dkk. (2023), mengkaji penerapan kode Reed Solomon $RS(15,9)$ dalam kriptosistem McEliece untuk melindungi transmisi pesan dari serangan komputer kuantum, dengan koreksi hingga 3 kesalahan bit menggunakan *galois field* $GF(2^4)$, menunjukkan bahwa pendekatan ini efektif dalam mempertahankan keamanan data.

Berdasarkan literatur pada penelitian sebelumnya, tujuan dari penelitian ini adalah untuk mengimplementasikan kode Reed-Solomon dalam mendeteksi serta mengoreksi kesalahan transmisi ayat Al-Qur’an berbasis huruf hijaiyah. Proses implementasi akan dilakukan menggunakan perangkat lunak SageMath. Melalui penerapan metode ini, diharapkan dapat meningkatkan akurasi dan ketelitian dalam pengiriman penulisan huruf hijaiyah, sehingga kesalahan yang umum terjadi dapat teridentifikasi dan diperbaiki secara otomatis. Penelitian ini juga menjadi langkah awal dalam memanfaatkan teori pengkodean untuk menjaga kemurnian teks suci Al-Qur’an secara digital.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang yang telah disampaikan, rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana simulasi proses deteksi kesalahan transmisi ayat Al-Qur'an menggunakan kode Reed-Solomon?
2. Bagaimana simulasi proses koreksi kesalahan transmisi ayat Al-Qur'an menggunakan kode Reed-Solomon?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah di atas, maka tujuan penelitian yang akan dicapai adalah sebagai berikut:

1. Untuk mendeskripsikan proses deteksi kesalahan transmisi ayat Al-Qur'an menggunakan kode Reed-Solomon.
2. Untuk mendeskripsikan proses koreksi kesalahan transmisi ayat Al-Qur'an menggunakan kode Reed-Solomon.

1.4 Manfaat Penelitian

Berdasarkan tujuan penelitian yang telah dikemukakan, hasil penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Manfaat Teoritis:

   Penelitian ini diharapkan dapat memberikan kontribusi dalam penerapan teori kode Reed-Solomon dalam bidang pemrosesan teks digital, khususnya untuk mendeteksi dan mengoreksi kesalahan transmisi ayat-ayat Al-Qur'an. Penerapan ini difokuskan pada teks berbahasa Arab yang menggunakan huruf

hijaiyah, sehingga diharapkan dapat meningkatkan akurasi dan keandalan dalam penyimpanan maupun transmisi data teks Al-Qur’an secara digital.

2. Manfaat Praktis:
    a. Memberikan ilustrasi teknis dan simulasi bagaimana kode Reed-Solomon dapat dimanfaatkan untuk mendeteksi dan memperbaiki kesalahan transmisi ayat Al-Qur’an secara otomatis.
    b. Menjadi rujukan atau dasar bagi penelitian selanjutnya dalam pengembangan sistem koreksi teks digital yang berbasis pada pengkodean huruf hijaiyah, baik untuk teks keagamaan maupun aplikasi teks Arab lainnya.
    c. Mendukung upaya pelestarian keakuratan dan kemurnian tulisan ayat-ayat Al-Qur’an dalam bentuk digital melalui pendekatan matematis dan teknologi informatika, sehingga dapat meningkatkan keandalan dalam distribusi dan penyimpanan data teks Al-Qur’an.

1.5 Batasan Masalah

Agar penelitian ini tidak keluar dari batas kajian yang ditetapkan, maka ruang lingkup penelitian dibatasi pada hal-hal berikut:

1. Penelitian ini menggunakan teks ayat Al-Qur'an dengan huruf hijaiyah dasar tanpa disertai:
    a. Harakat (tanda vokal seperti fathah, kasrah, dammah, dan sebagainya).
    b. Tanda baca (tanda waqaf, nomor ayat, rukuk, dan lainnya).
    c. Bentuk huruf yang berbeda (awal, tengah, akhir) tetap direpresentasikan menggunakan *Unicode* yang sama, kecuali huruf khusus yang memang

memiliki *Unicode* berbeda seperti berbagai bentuk hamzah, *Tā' Marbūṭah* (ۃ), *Alif Maqṣūrah* (ى) dan spasi.

2. Setiap huruf hijaiyah direpresentasikan dalam bentuk kode *Unicode* 16-bit, khususnya dalam rentang *Unicode Block Arabic* ($UTF-8$).
3. Sumber penulisan ayat Al-Qur'an.
    a. Penelitian menggunakan ayat Al-Qur'an dalam bentuk digital, seperti mushaf elektronik, aplikasi web, atau dokumen digital lainnya.
    b. Penelitian mengikuti standar khat Naskhi yang digunakan dalam mushaf Rasm Utsmani cetakan Indonesia, serta tidak membahas perbedaan bentuk tulisan pada mushaf versi internasional.
    c. Data yang digunakan adalah sepuluh penulisan ayat Al-Qur'an sebagai data uji, di mana kesalahan simbol ditambahkan secara sengaja dan terkontrol untuk mensimulasikan kondisi transmisi digital.
4. Penelitian ini difokuskan pada objek kesalahan sebagai berikut:
    a. Kesalahan yang timbul akibat ketidaksesuaian penulisan huruf hijaiyah dalam aplikasi digital, situs web**,** atau sistem input teks.
    b. Penyimpangan atau penyelewengan penulisan ayat, baik disengaja maupun tidak disengaja, yang menyebabkan perubahan struktur teks huruf hijaiyah.
    c. Model kesalahan dalam penelitian ini merupakan kesalahan yang sengaja disisipkan (*injected errors*) selama proses transmisi digital**,** bukan kesalahan dari kanal komunikasi nyata. Tujuannya adalah untuk menguji kemampuan deteksi dan koreksi dari kode Reed-Solomon secara terkontrol.

d. Setiap kesalahan penulisan huruf hijaiyah direpresentasikan sebagai *symbol error* dalam proses transmisi pada kode Reed-Solomon.

5. Penelitian ini tidak membahas aspek tajwid, tafsir, maupun qira'at (variasi bacaan), serta tidak mencakup keseluruhan sistem Rasm Utsmani secara komprehensif.
6. Batasan pada Implementasi Reed-Solomon.
   a. Penelitian ini hanya menguji kemampuan deteksi dan koreksi kesalahan (*error*) pada level simbol (huruf hijaiyah yang diubah menjadi simbol 16-bit).
   b. Penelitian tidak membahas optimasi algoritma atau implementasi Reed–Solomon untuk skala besar, melainkan hanya menerapkan RS dengan parameter yang digunakan dalam studi kasus.
   c. Pengujian dilakukan dalam lingkungan simulasi perangkat lunak SageMath bukan pada sistem transmisi fisik sebenarnya.

1.6 Definisi Istilah

Terdapat beberapa istilah yang digunakan dalam penelitian ini, yakni sebagai berikut:

Unicode	:	Standar pengkodean karakter yang memberikan kode unik untuk setiap huruf, angka, dan simbol dari berbagai bahasa di dunia.
Coding	:	Proses pemberian kode atau representasi tertentu terhadap data atau informasi ke dalam bentuk simbol, bilangan, atau rangkaian bit agar

dapat diproses, disimpan, atau ditransmisikan oleh suatu sistem.

Encoding	:	*Encoding* adalah metode dalam teori pengkodean yang berfungsi mengubah data asli menjadi bentuk kode tertentu, biasanya dalam bentuk simbol, kode, atau format digital, agar dapat disimpan, diproses, atau dikirim secara lebih efisien.
Decoding	:	*Decoding* adalah proses kebalikan dari *encoding* yaitu metode dalam teori koding yang mengubah kembali informasi yang telah dikodekan tersebut menjadi data asli.
Deteksi	:	Deteksi adalah usaha untuk menemukan, mengenali, dan menentukan keberadaan suatu kesalahan atau penyimpangan.
Koreksi	:	Koreksi adalah proses perbaikan atau pembetulan terhadap suatu kesalahan yang ditemukan agar sesuai dengan bentuk yang benar.
Teori Pengkodean	:	Teori pengkodean adalah cabang ilmu dalam matematika terapan dan teknik informatika yang mempelajari cara merepresentasikan informasi ke dalam bentuk simbol atau kode tertentu

		dengan tujuan meningkatkan keandalan dan efisiensi komunikasi.
Kode Reed-Solomon	:	Reed Solomon adalah sebuah metode dalam teori kode error yang digunakan untuk mendeteksi dan memperbaiki kesalahan dalam transmisi data.
Shortened Code Reed-Solomon	:	Kode Reed–Solomon yang dipendekkan, yaitu teknik memperpendek panjang codeword dengan menambahkan simbol isian (padding) pada awal pesan, kemudian membuangnya setelah proses pengkodean.
Codeword	:	Hasil *encoding* berupa data yang telah diubah menjadi bentuk vektor untuk proses *decoding*.
Padding	:	Proses menambahkan data tambahan (biasanya berupa bit tertentu) ke suatu pesan agar pesan tersebut memenuhi ukuran atau panjang yang diperlukan oleh suatu algoritma, sistem, atau blok data.
SageMath	:	Perangkat lunak *open-source* berbasis Python yang digunakan untuk komputasi matematika, termasuk aljabar, kriptografi, teori bilangan, dan teori pengkodean.
Transmisi	:	Pengiriman (penerusan) pesan dan sebagainya dari seseorang kepada orang (benda) lain.

Transmisi Data	:	Proses pengiriman data dari satu sumber ke penerima data.
Transmisi Ayat Al-Qur’an	:	Transmisi ayat Al-Qur’an adalah proses pengiriman atau penyampaian teks ayat dari satu media atau perangkat ke media lain secara digital.
Kesalahan (*error*)	:	Hal yang terjadi apabila suatu hal tidak bertindak semestinya, seperti salah sasaran, kehilangan satu bit, atau juga berubah datanya.

BAB II
KAJIAN TEORI

2.1 Lapangan

Definisi 2.1

Lapangan adalah suatu ring komutatif yang memiliki elemen identitas, di mana setiap elemen yang bukan nol adalah suatu unit (mempunyai invers terhadap perkalian) (Gallian, 2021).

Definisi 2.2

Lapangan F merupakan himpunan elemen-elemen tertutup yang memuat dua operasi biner yaitu penjumlahan dan perkalian dinotasikan dengan "$+$" dan "$\cdot$" sehingga aksioma-aksioma di bawah ini terpenuhi untuk semua $a, b, c \in F$ (Menezes dkk., 1996).

1. $a + (b + c) = (a + b) + c$
2. $a + b = b + a$
3. Terdapat elemen identitas $0 \in F$ sedemikian sehingga $a + 0 = a$
4. Terdapat elemen $-a \in F$ sedemikian sehingga $a + (-a) = 0$
5. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
6. $a \cdot b = b \cdot a$
7. Terdapat elemen $1 \in F$ sedemikian sehingga $a \cdot 1 = a$
8. untuk setiap $a \neq 0$, terdapat elemen $a^{-1} \in F$ sedemikian sehingga $a \cdot a^{-1} = 1$
9. $a \cdot (b + c) = a \cdot b + a \cdot c$.

Contoh 2.3

Himpunan $\mathbb{Z}_5 = \{[0], [1], [2], [3]. [4]\}$ merupakan himpunan semua kelas sisa bilangan bulat modulo 5. Dengan $+_5$ dan $*_5$ pada himpunan $\mathbb{Z}_5$ membentuk suatu lapangan.

Bukti:

Tabel 2.1 Operasi Penjumlahan Modulo 5 di $\mathbb{Z}_5$

$+_5$	[0]	[1]	[2]	[3]	[4]
[0]	[0]	[1]	[2]	[3]	[4]
[1]	[1]	[2]	[3]	[4]	[0]
[2]	[2]	[3]	[4]	[0]	[1]
[3]	[3]	[4]	[0]	[1]	[2]
[4]	[4]	[0]	[1]	[2]	[3]

Tabel 2.2 Operasi Perkalian Modulo 5 di $\mathbb{Z}_5$

$*_5$	[0]	[1]	[2]	[3]	[4]
[0]	[0]	[0]	[0]	[0]	[0]
[1]	[0]	[1]	[2]	[3]	[4]
[2]	[0]	[2]	[4]	[1]	[3]
[3]	[0]	[3]	[1]	[4]	[2]
[4]	[0]	[4]	[3]	[2]	[1]

Dengan memperhatikan Tabel 2.1 untuk operasi $+_5$ sifat tertutup terpenuhi, karena untuk setiap $[a], [b] \in \mathbb{Z}_5$, berlaku $[a]+_5[b] \in \mathbb{Z}_5$. Elemen identitas terhadap penjumlahan adalah [0], dan setiap elemen $[a] \in \mathbb{Z}_5$ juga memiliki invers penjumlahan yang dinyatakan dengan $-[a]$, sehingga berlaku $(-[a]+_5[a]) = [a]+_5(-[a]) = [0]$. Selain itu, tabel penjumlahan bersifat simetris terhadap diagonal utama, sehingga operasi $+_5$ bersifat komutatif, yaitu $[a]+_5[b] = [b]+_5 [a], \ \forall [a], [b] \in \mathbb{Z}_5$.

Selanjutnya, berdasarkan Tabel 2.2 untuk operasi $*_5$, himpunan $\mathbb{Z}_5$ bersifat tertutup terhadap perkalian., karena untuk setiap $[a], [b] \in \mathbb{Z}_5$ berlaku $[a] *_5 [b] \in \mathbb{Z}_5$. Elemen identitas terhadap perkalian adalah $[1]$, dan setiap elemen tak nol $[a] \in \mathbb{Z}_5$ memiliki invers perkalian, yaitu elemen $[a]^{-1}$ yang memenuhi $[a] *_5 [a]^{-1} = [a]^{-1} *_5 [a] = [1]$. Tabel perkalian juga simetris terhadap diagonal utama, sehingga operasi $*_5$ bersifat komutatif. Karena $\mathbb{Z}_5$ memenuhi sifat tertutup, memiliki elemen identitas, setiap elemen memiliki invers terhadap operasi penjumlahan dan perkalian, serta kedua operasi bersifat komutatif dan asosiatif, maka dapat disimpulkan bahwa $(\mathbb{Z}_5, +_5, *_5)$ merupakan suatu lapangan (*field*).

2.1.1 Lapangan Hingga (*Galois Field*)

Definisi 2.3

Suatu lapangan yang memiliki elemen sebanyak berhingga disebut dengan *galois field* "lapangan berhingga" (Dummit & Foote, 2004).

Jika suatu lapangan memiliki p^n elemen, dengan p adalah bilangan prima dan $n \in \mathbb{Z}^+$. Maka lapangan hingga dinotasikan dengan $GF(q)$ atau F_q, dengan $q = p^n$. Untuk $n = 1$, elemen lapangan $GF(p)$ sama dengan $\{0,1,2, \dots, p-1\}$, dengan operasi penjumlahan $(+)$ dan perkalian $(\cdot)$ dilakukan secara modulo p. Sebagai contoh, $GF(2)$ memiliki 2 elemen $\{0,1\}$, sedangkan $GF(3)$ memiliki 3 elemen $\{0,1,2\}$. Jika $n > 1$ maka lapangan $GF(p^n)$ dibangun dari polinomial irreduksibel derajat n atas $GF(p^n)$. Misalnya, $GF(2^2)$ dapat dibentuk dari $\mathbb{Z}_2[x]/p(x)$, dengan $p(x) = x^2 + x + 1$ sehingga memiliki 4 elemen berbeda.

2.1.2 Aritmatika pada *Galois Field*

Dalam *galois field*, operasi penjumlahan, pengurangan, perkalian, dan pembagian dapat dilakukan, namun berbeda dengan operasi pada bilangan real. Setiap hasil operasi antar elemen dalam *galois field* selalu menghasilkan elemen lain yang masih berada dalam himpunan terbatas tersebut.

1. Operasi Penjumlahan dan Pengurangan.

   Cara melakukan operasi penjumlahan adalah sebagai berikut, $(a_m x^m + \cdots + a_1 x + a_0) + (b_m x^m + \cdots + b_1 x + b_0) = c_m x^m + \cdots + c_1 x + c_0$, dimana $c_i = a_i + b_i$ untuk $(m-1) \geq i \geq 0$. Sama halnya dalam operasi pengurangan dua elemen $GF(a_m x^m + \cdots + a_1 x + a_0) - (b_m x^m + \cdots + b_1 x + b_0) = c_m x^m + \cdots + c_1 x + c_0$, dimana $c_i = a_i + b_i$ untuk $(m-1) \geq i \geq 0$. Operasi $+$ dan $-$ dilakukan dengan modulo 2. Sehingga, baik penjumlahan maupun pengurangan akan didapatkan hasil $c_i = 0$ untuk $a_i = b_i$ dan $c_i = 1$ untuk $a_i \neq b_i$. Dengan kata lain, penjumlahan dan pengurangan adalah identik dalam *Galois Field*.

   Sebagai contoh, pada $GF(16)$ kita dapat mengurangkan $\alpha^3 + \alpha$ dengan $\alpha^3 + \alpha^2 + \alpha + 1$, sehingga $(\alpha^3 + \alpha) - (\alpha^3 + \alpha^2 + \alpha + 1) = \alpha^2 + 1$. Dalam representasi biner, $(1010) - (1111) = (1010) + (1111) = 0101$. Ataupun dalam representasi desimal, $10 - 15 = 10 + 15 = 5$.

2. Operasi Perkalian dan Pembagian.

   Dengan *primitive* polinomial dapat ditentukan bentuk tiap elemen bilangan GF, baik dalam bentuk polinomial, biner, ataupun desimal. Dengan mengetahui bentuk elemen bilangan tersebut, dapat dicari hasil operasi perkalian dua elemen bilangan GF. Pada $GF(2^m)$ perkalian dua elemen

bilangan $a(x) = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_1 x^1 + a_0$ dan $b(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_1 x^1 + b_0$ akan menghasilkan $c(x) = c_m x^m + c_{m-1} x^{m-1} + \cdots + c_1 x^1 + c_0$, dimana konstanta $c_m, \ldots, c_0$ didapatkan dari reduksi bilangan berdasarkan *primitive polynomial*.

Sebagai contoh, operasi perkalian antara bilangan 14 dengan 11 pada $GF(2^4)$. Representasi hasil perkalian dalam bentuk *index* adalah $\alpha^{(\text{jumlah kedua } index) mod (2^{m-1})}$. Bilangan 14 merepresentasikan α^{11} dan bilangan 11 merepresentasikan α^7. Sehingga, $\alpha^{11} \times \alpha^7 = \alpha^{18 mod (15)} = \alpha^3 = 8$, atau $14 \times 11 = 8$ pada $GF(2^4)$.

Operasi pembagian 8 dengan 14 pada $GF(2^4)$ mirip dengan perkalian. Representasi hasil pembagian dalam bentuk indeks adalah $\alpha^{(\text{selisih kedua } index) mod (2^{m-1})}$. Bilangan 8 merepresentasikan α^3 dan bilangan 14 merepresentasikan α^{11}. Sehingga, $\alpha^3 / \alpha^{11} = \alpha^{(3-11) mod^{15}} = \alpha^{-(8) mod (15)} = \alpha^7 = 11$. Elemen invers dari GF didefinisikan sebagai nilai elemen bilangan yang jika dikalikan menghasilkan nilai 1. Jika 14 merepresentasikan α^{11} maka inversnya adalah $\alpha^{-11} = \alpha^{(-11) \text{mod} (15)} = \alpha^4 = 3$. Maka, $8/14 = 8\ x\ 3 = \alpha^3 \times \alpha^4 = \alpha^{7 mod (15)} = \alpha^7 = 11$, atau $8/14 = 11$.

3. *Exclusive*-*OR* (XOR).

   Pada pengolahan data digital, salah satu operasi biner yang paling mendasar dan sering digunakan dalam proses *encoding* dan *decoding* adalah operasi *Exclusive*-*OR* (XOR), yang dilambangkan dengan simbol " $\oplus$ ". Operasi XOR merupakan bentuk penjumlahan dalam modulo 2 pada lapangan hingga $GF(2)$, yaitu suatu himpunan bilangan biner yang hanya terdiri dari

dua elemen, yaitu 0 dan 1. Secara matematis, operasi XOR didefinisikan sebagai $a \oplus b = (a + b) mod 2$. Operasi ini mendefinisikan bahwa dua bit yang berbeda akan menghasilkan nilai 1, sedangkan dua bit yang bernilai sama menghasilkan nilai 0 (Makhomah dkk., 2021).

Misalkan a, b, c merupakan peubah Boolean dalam $GF(2)$, maka operasi XOR memiliki beberapa sifat penting sebagai berikut:

a. $a \oplus a = 0$ (Sifat invers penjumlahan),

b. $a \oplus 0 = a$ (Sifat identitas),

c. $a \oplus b = b \oplus a$ (Sifat komutatif),

d. $a \oplus (b \oplus c) = (a \oplus b) \oplus$ c (Sifat asosiatif).

Contoh 2.1

a. $1 \oplus 1 = (1 + 1) \bmod 2 = 0$

b. $0 \oplus 0 = (0 + 0) \bmod 2 = 0$

c. $1 \oplus 0 = 0 \oplus 1 \ = \ 1$

d. $1 \oplus (0 \oplus 1) = (1 \oplus 0) \oplus 1 = 0$

Jika dua bilangan biner dioperasikan menggunakan XOR, maka operasi dilakukan dengan menerapkan XOR pada setiap pasangan bit yang bersesuaian.

Contoh 2.2

$$10011 \oplus 11001 = 01010$$

pada hal ini, hasilnya diperoleh sebagai berikut:

$$\begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ \hline 1 \oplus 1 & 0 \oplus 1 & 0 \oplus 0 & 1 \oplus 0 & 1 \oplus 1 \\ 0 & 1 & 0 & 1 & 0 \end{array} \oplus$$

2.1.3 Polinomial *Galois Field*

Sebuah polinomial $a(x)$ berderajat m pada lapangan hingga $GF(p)$ dituliskan sebagai,

$$a(x) = a_m x^m + a_{m-1} x^{m-1} + \ ... \ + a_1 x + a_0 \qquad (2.1)$$

dengan koefisien $a_i \in GF(p)$ dan $a_m \neq 0$. Derajat (*degree*) dari polinomial $a(x)$ adalah pangkat tertinggi dari x, yaitu m. Polinomial tersebut dapat dijumlahkan, dikurangkan, dikalikan, dan dibagi dengan polinomial pada *field* yang sama. Misalkan $n(x)$ dan $m(x)$ adalah polinomial pada $GF(256)$, maka $n(x) + m(x), n(x) - m(x), n(x) \times m(x)$, dan $n(x)/m(x)$ terdefinisi.

Lapangan hingga $GF(p^m)$ dapat direpresentasikan dengan suatu elemen primitif (*primitive element*) α, yaitu

$$GF(p^m) = \{0, \alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{(p^m-2)}\}.$$

Setiap elemen α^i dapat direpresentasikan dalam bentuk polinomial berderajat kurang dari m, yaitu

$$a_{m-1} x^{m-1} + \ ... \ + a_1 x + a_0,$$

dengan koefisien $\alpha_j \in \{0,1\}$. Representasi ini juga dapat dinyatakan dalam bentuk bilangan biner $m - bit$ maupun bentuk desimal. Dalam pengkodean Reed-Solomon, elemen primitif α umumnya dipilih dengan representasi desimal 2.

Definisi 2.5

Polinomial tak tereduksi $f(x)$ dengan derajat m atas $GF(q)$ dikatakan primitif jika n adalah bilangan bulat positif terkecil, untuk $f(x)$ faktor dari $x^n - 1$, berlaku $n = q^m - 1$.

Sebagai contoh, lapangan hingga $GF(8) = GF(2^3)$ memiliki $m = 3$ dan dapat direpresentasikan menggunaka polinomial primitif $p(x) = x^3 + x + 1$. Jika α adalah akar dari $p(x)$, maka berlaku,

$$\alpha^3 + \alpha + 1 = 0 \Rightarrow \alpha^3 = \alpha + 1.$$

Setiap elemen di $GF(8)$ dapat direpresentasikan sebagai $\alpha_2 x^2 + \alpha_1 x^1 + \alpha_0 x^0,\ \alpha_j \in \{0,1\}$. Dengan $\alpha_2, \alpha_1, \alpha_0$ berkorespondensi dengan nilai biner 000 sampai 111, atau dalam bentuk desimal 0 sampai 7 (Wicker, 2005).

Tabel 2.3 Tabel Representasi Eksponensial dan Polinomial Elemen $GF(2^n)$

Representasi Eksponensial	Representasi Polinomial
1	1
α^1	α
α^2	α^2
α^3	$\alpha^3 = \alpha + 1$
α^4	$\alpha(\alpha + 1) = \alpha^2 + \alpha$
α^5	$\alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$
α^6	$\alpha^3 + \alpha^2 + \alpha = \alpha^2 + 1$
0	0

Sebuah polinomial pada *galois field* yang dituliskan sebagai $p(x)$ disebut *primitive polinomial*. Polinomial ini digunakan untuk membangkitkan seluruh elemen dalam lapangan hingga melalui proses perkalian antar elemen. Dalam *galois field* dengan ukuran tertentu, bentuk polinomial primitif yang umum digunakan dapat dilihat pada Tabel 2.4.

Tabel 2.4 Tabel Primitif Polinomial *GF*

m	*Primitive polynomial GF*	m	*Primitive polynomial GF*
2	$1 + x + x^2$	13	$1 + x + x^3 + x^4 + x^{13}$
3	$1 + x + x^3, 1 + x^2 + x^3$	14	$1 + x + x^6 + x^{10} + x^{14}$
4	$1 + x + x^4$	15	$1 + x + x^{15}$
5	$1 + x^2 + x^5$	16	$1 + x + x^3 + x^{12} + x^{16}$
6	$1 + x + x^6$	17	$1 + x^3 + x^{17}$
7	$1 + x^3 + x^7$	18	$1 + x^7 + x^{18}$
8	$1 + x^2 + x^3 + x^4 + x^8$	19	$1 + x + x^2 + x^5 + x^{19}$
9	$1 + x^4 + x^9$	20	$1 + x^3 + x^{20}$
10	$1 + x^3 + x^{10}$	21	$1 + x^2 + x^{21}$
11	$1 + x^2 + x^{11}$	22	$1 + x + x^{22}$
12	$1 + x + x^4 + x^6 + x^{12}$	23	$1 + x + x^{22}$

2.1.4 Representasi *Galois Field*

Seperti yang dijelaskan sebelumnya, *galois field* direpresentasikan sebagai himpunan terbatas dengan jumlah elemen maksimum sebesar 2^m. Pada kasus $GF(16)$ dengan $m = 4$, jumlah elemennya adalah $2^4 = 16$. Untuk membangunnya, diperlukan sebuah polinomial primitif $p(x) = x^4 + x + 1$, maka substitusi $p(x) = 0$ dihasilkan $\alpha^4 = \alpha + 1$.

Relasi ini sangat penting karena digunakan untuk menyederhanakan nilai-nilai pangkat α^i yang lebih tinggi dengan cara mengganti α^4 dan kelipatannya menggunakan $\alpha + 1$. Kemudian melakukan operasi penjumlahan polinomial dalam $GF(2)$. Proses ini dilakukan berulang hingga seluruh elemen dalam $GF(16)$ terbentuk. Sehingga diperoleh $GF(16) = \{0,1, \alpha, \alpha^2, \alpha^3, \alpha^4, \dots, \alpha^{14}\}$.

Tabel 2.5 menunjukkan representasi elemen $GF(16)$ dalam bentuk pangkat α^i, polinomial, biner 4-bit, dan desimal.

Tabel 2.5 Representasi Polinomial, Biner, dan Desimal pada $GF(2^4)$

α^i	Representasi Polinomial	Biner	Desimal
0	0	0000	0
1	1	0001	1
α^1	α	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	$\alpha + 1$	0011	3
α^5	$\alpha^2 + \alpha$	0110	6
α^6	$\alpha^3 + \alpha^2$	1100	12
α^7	$\alpha^3 + \alpha + 1$	1011	11
α^8	$\alpha^2 + 1$	0101	5
α^9	$\alpha^3 + \alpha$	1010	10
α^{10}	$\alpha^2 + \alpha + 1$	0111	7
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	1110	14
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	1111	15
α^{13}	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$	1101	13
α^{14}	$\alpha^3 + 1$	1001	9

Sementara itu, untuk $GF(256)$ dengan $m = 8$, jumlah elemennya adalah $GF(256) = 28 = 256$. Dengan memilih polinomial primitif misalnya $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. Maka diperoleh $\alpha^8 = \alpha^4 + \alpha^3 + \alpha^2 + 1$. Representasi elemen $GF(256)$ juga dapat disajikan dalam bentuk pangkat α^i, polinomial, biner 8-bit, dan desimal pada Tabel 2.6.

Tabel 2.6 Representasi Polinomial, Biner, dan Desimal pada $GF(2^8)$

α^i	Representasi Polinomial	Biner	Desimal
α^0	α^0	00000001	1
α^1	α^1	00000010	2
α^2	α^2	00000100	4
α^3	α^3	00001000	8
α^4	α^4	00010000	16
α^5	α^5	00100000	32
α^6	α^6	01000000	64
α^7	α^7	10000000	128
α^8	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha^0$	00011101	29
α^9	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	00111010	58
α^{10}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2$	01110100	116
α^{11}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3$	11101000	232
α^{12}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^0$	11001101	205
α^{13}	$\alpha^7 + \alpha^2 + \alpha^1 + \alpha^0$	10000111	135
α^{14}	$\alpha^4 + \alpha^1 + \alpha^0$	00010011	19
α^{15}	$\alpha^5 + \alpha^2 + \alpha^1$	00100110	38
⋮	⋮	⋮	⋮
α^{255}	α^0	00000001	1

2.1.5 Ruang Vektor atas Lapangan Hingga

Suatu ruang vektor V atas lapangan F adalah himpunan tak kosong yang dilengkapi dengan dua operasi, yaitu penjumlahan vektor dan perkalian dengan skalar (Lang dkk., 2015). Untuk setiap vektor $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \in V$ dan skalar $\lambda, \mu \in F$, berlaku aksioma-aksioma vektor berikut:

1. $\boldsymbol{u} + \boldsymbol{v} \in V$.
2. $(\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w} = \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w})$.

3. Terdapat $\mathbf{0} \in V$ untuk setiap $\boldsymbol{v} \in V$ sedemikian sehingga memenuhi $\mathbf{0} + \boldsymbol{v} = \boldsymbol{v} + \mathbf{0} = \boldsymbol{v}$.
4. Untuk setiap $\boldsymbol{u} \in V$ terdapat $-\boldsymbol{u} \in V$ sedemikian sehingga memenuhi $\boldsymbol{u} + (-\boldsymbol{u}) = (-\boldsymbol{u}) + \boldsymbol{u} = \mathbf{0}$.
5. $\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}$.
6. $\lambda \boldsymbol{v} \in V$.
7. $\lambda(\boldsymbol{u} + \boldsymbol{v}) = \lambda \boldsymbol{u} + \lambda \boldsymbol{v}$.
8. $(\lambda + \mu)\boldsymbol{u} = \lambda \boldsymbol{u} + \mu \boldsymbol{u}$.
9. $(\lambda \mu)\boldsymbol{u} = \lambda(\mu \boldsymbol{u})$.
10. $1\boldsymbol{u} = \boldsymbol{u}$.

Ruang vektor digunakan untuk merepresentasikan pesan dan kode sebagai vektor berdimensi tetap atas lapangan hingga $GF(p^n)$. Proses *encoding* dengan menggunakan matriks generator merupakan bentuk transformasi linier dalam ruang vektor,

di mana:

1. Pesan berupa vektor $m \in GF(p^n)^k$.
2. Dikalikan dengan matriks generator $G \in GF(p^n)^{k \times n}$.
3. Menghasilkan kode $C = mG \in GF(p^n)$.

2.2 Teori Pengkodean (*Coding Theory*)

Teori pengkodean merupakan cabang ilmu matematika terapan yang mempelajari cara merepresentasikan informasi dalam bentuk kode sehingga data dapat ditransmisikan atau disimpan secara efisien dan tetap tahan terhadap gangguan (*noise*). Ketika data dikirimkan melalui saluran komunikasi,

kemungkinan terjadinya gangguan (*noise*) sangat besar sehingga data yang diterima dapat berbeda dari yang dikirim. Fokus utama teori pengkodean adalah merancang kode yang memungkinkan deteksi dan koreksi kesalahan, sehingga informasi dapat dipulihkan meskipun terjadi kerusakan data selama proses pengiriman.

Secara umum, teori pengkodean menggabungkan konsep-konsep matematika seperti aljabar linear, teori bilangan, polinomial, dan lapangan hingga (*Galois Field*). Dalam konteks transmisi digital, suatu kode dibangun untuk mengubah pesan asli ke dalam bentuk *codeword* yang memiliki redundansi tertentu. Redundansi inilah yang memungkinkan penerima mendeteksi dan memperbaiki kesalahan yang terjadi selama proses transmisi.

Teori koding digunakan untuk mengkoreksi kesalahan pada saluran informasi yang apabila terjadi gangguan dapat membuat data tidak terkirim sempurna. *Encoding* atau enkripsi adalah suatu cara atau metode dalam teori koding yang mengubah suatu data asli menjadi kode-kode yang melambangkan data tersebut. *Decoding* atau dekripsi merupakan suatu proses kebalikan dari *encoding*. Pengertian *decoding* yaitu suatu cara atau metode dalam teori koding yang mengubah kode-kode data tersebut menjadi data asli. (Munir, 2006).

2.2.1 Kode Siklik

Sebuah kode linier C dikatakan siklik jika untuk setiap vektor $c = (c_1, c_2, \dots, c_{n-1}) \in C$. Contohnya, jika $(1,1,0,1)$ adalah elemen sebuah kode siklik, maka $(1,1,1,0)$ juga termuat dalam kode siklik tersebut (Jamal dkk., 2012). Dengan demikian, operasi pergeseran siklik memetakan kode C ke dirinya sendiri. Sehingga jika diberikan suatu matriks G yang merentang kode siklik, untuk

menentukan semua vektor kode dari kode sikliknya dapat dilakukan dengan melakukan pergeseran secara siklik pada vektor perentangnya.

2.2.2 Kode Linear

Definisi 2.6

Suatu kode linear C dengan panjang n atas F_q adalah subruang dari F_n^q. Dimensi dari kode linear C adalah dimensi dari C sebagai ruang vektor atas F_q yang dinotasikan dim C (Bierbrauer, 2019).

$$F_q^n = \{(v_1, v_2, \dots, v_n) \mid v_i \in F_q\} \tag{2.2}$$

di mana:

F_q	:	Lapangan hingga dengan q elemen
F_q^n	:	Ruang vektor berdimensi n atas F_q
v_i	:	Elemen ke-i dari vektor
$(v_1, v_2, \dots, v_n)$	:	Vektor berdimensi n

Kode linear dengan panjang n dan dimensi k disebut sebagai kode (n, k). Artinya, kode linear C terdiri atas q^k *codeword*, karena terdapat q^k kombinasi linear yang dapat dibentuk dari basis berdimensi k di F_q^n.

1. Matriks Generator.

   Pada teori pengkodean, sebuah kode linear $C[n, k]$ atas lapangan F_q dapat direpresentasikan menggunakan matriks generator G berukuran $k \times n$. Baris-baris pada matriks ini membentuk basis dari subruang $C \subseteq F_q^n$. Dengan demikian, setiap *codeword* $c \in C$ dapat diperoleh dari hasil perkalian antara vektor pesan $v \in F_q^k$ dengan matriks G, yaitu:

$$c = v \cdot G \tag{2.3}$$

Matriks generator G dinotasikan dengan:

$$G = [I_k \mid A] \tag{2.4}$$

di mana:

I_k : Matriks identitas berukuran $k \times k$

A : Matriks berukuran $k \times (n-k)$

Misalkan, diberikan dua vektor pesan sebagai berikut:

$$v_1 = (1011), \quad v_2 = (0101)$$

dan matriks generator G sebagai berikut:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Dengan demikian, diperoleh:

$$c_1 = v_1 \cdot G = (1\,0\,1\,1\,0\,0\,0)$$

$$c_2 = v_2 \cdot G = (0\,1\,0\,1\,1\,0\,1)$$

Untuk kode Reed–Solomon (RS), matriks generator tidak dibentuk dari $G = [I_k|A]$ secara langsung, tetapi melalui konstruk *Vandermonde matrix* di lapangan hingga $GF(2^m)$. Matriks generator RS berukuran $k \times n$ umumnya dituliskan sebagai:

$$G = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_{n-1}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^{k-1} & x_1^{k-1} & x_2^{k-1} & \cdots & x_{n-1}^{k-1} \end{bmatrix}$$

Dengan $x_j = \alpha^{5j}$ merupakan titik evaluasi yang berbeda pada $GF(2^m)$, dan $j = 0,1,\ldots,n-1$ menyatakan indeks kolom matriks generator.

Dalam penelitian ini titik evaluasi dipilih menggunakan pola aritmatika dengan parameter *start*$= 0$ dan *step*$= 5$, dan titik evaluasi menjadi:

$$x_j = \alpha^{5j}.$$

Dengan demikian, kolom ke-j dari matriks generator adalah:

$$\left[1,\ \alpha^{5j},\ \alpha^{10j},\ \alpha^{15j}, \ldots, \alpha^{k-1(5j)}\right]^T.$$

Karena pada $GF(2^8)$ berlaku $\alpha^{255} = \alpha^0 = 1$, semua eksponen dipandang modulo 255. Dengan $gcd(5{,}255) = 5$, deret $5j\ (mod 255)$ akan mengulangi setiap $255/5 = 51$ langkah, oleh karena itu jumlah kolom unik maksimum tanpa pengulangan eksponen adalah 51. Untuk menjamin bahwa semua titik evaluasi berbeda (sehingga submatriks Vandermonde $k \times k$ nonsingular), dipastikan $n \leq 51$ saat menggunakan pola $start = 0$, $step = 5$ pada $GF(2^8)$.

$$G = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \alpha^{5} & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \alpha^{65} & \alpha^{70} & \alpha^{75} & \cdots & \alpha^{250} \\
1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \alpha^{130} & \alpha^{140} & \alpha^{150} & \cdots & \alpha^{245} \\
1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \alpha^{195} & \alpha^{210} & \alpha^{222} & \cdots & \alpha^{240} \\
1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{140} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \alpha^{5} & \alpha^{25} & \alpha^{45} & \cdots & \alpha^{235} \\
1 & \alpha^{25} & \alpha^{50} & \alpha^{75} & \alpha^{100} & \alpha^{125} & \alpha^{150} & \alpha^{175} & \alpha^{200} & \alpha^{225} & \alpha^{250} & \alpha^{20} & \alpha^{45} & \alpha^{70} & \alpha^{95} & \alpha^{120} & \cdots & \alpha^{230} \\
1 & \alpha^{30} & \alpha^{60} & \alpha^{90} & \alpha^{120} & \alpha^{150} & \alpha^{180} & \alpha^{210} & \alpha^{240} & \alpha^{15} & \alpha^{45} & \alpha^{75} & \alpha^{105} & \alpha^{135} & \alpha^{165} & \alpha^{195} & \cdots & \alpha^{225} \\
1 & \alpha^{35} & \alpha^{70} & \alpha^{105} & \alpha^{140} & \alpha^{175} & \alpha^{210} & \alpha^{245} & \alpha^{25} & \alpha^{60} & \alpha^{95} & \alpha^{130} & \alpha^{165} & \alpha^{200} & \alpha^{235} & \alpha^{15} & \cdots & \alpha^{220} \\
1 & \alpha^{40} & \alpha^{80} & \alpha^{120} & \alpha^{160} & \alpha^{200} & \alpha^{240} & \alpha^{25} & \alpha^{65} & \alpha^{105} & \alpha^{9} & \alpha^{185} & \alpha^{225} & \alpha^{10} & \alpha^{50} & \alpha^{90} & \cdots & \alpha^{215} \\
1 & \alpha^{45} & \alpha^{90} & \alpha^{135} & \alpha^{180} & \alpha^{225} & \alpha^{15} & \alpha^{60} & \alpha^{105} & \alpha^{150} & \alpha^{145} & \alpha^{240} & \alpha^{30} & \alpha^{75} & \alpha^{120} & \alpha^{165} & \cdots & \alpha^{210} \\
1 & \alpha^{50} & \alpha^{100} & \alpha^{150} & \alpha^{200} & \alpha^{250} & \alpha^{45} & \alpha^{95} & \alpha^{145} & \alpha^{195} & \alpha^{195} & \alpha^{40} & \alpha^{90} & \alpha^{140} & \alpha^{190} & \alpha^{240} & \cdots & \alpha^{205} \\
1 & \alpha^{55} & \alpha^{110} & \alpha^{165} & \alpha^{220} & \alpha^{20} & \alpha^{75} & \alpha^{130} & \alpha^{185} & \alpha^{240} & \alpha^{245} & \alpha^{95} & \alpha^{150} & \alpha^{205} & \alpha^{5} & \alpha^{60} & \cdots & \alpha^{200} \\
1 & \alpha^{60} & \alpha^{120} & \alpha^{180} & \alpha^{240} & \alpha^{45} & \alpha^{105} & \alpha^{165} & \alpha^{225} & \alpha^{30} & \alpha^{40} & \alpha^{150} & \alpha^{210} & \alpha^{15} & \alpha^{75} & \alpha^{135} & \cdots & \alpha^{195} \\
1 & \alpha^{65} & \alpha^{130} & \alpha^{195} & \alpha^{5} & \alpha^{70} & \alpha^{135} & \alpha^{200} & \alpha^{10} & \alpha^{75} & \alpha^{90} & \alpha^{205} & \alpha^{15} & \alpha^{80} & \alpha^{145} & \alpha^{210} & \cdots & \alpha^{190} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha^{230} & \alpha^{205} & \alpha^{180} & \alpha^{155} & \alpha^{130} & \alpha^{105} & \alpha^{80} & \alpha^{55} & \alpha^{30} & \alpha^{5} & \alpha^{235} & \alpha^{210} & \alpha^{185} & \alpha^{160} & \alpha^{135} & \cdots & \alpha^{25}
\end{bmatrix}$$

2. Matriks *Parity-Check.*

Setiap kode linear $C[n, k]$ atas lapangan F_q, terdapat matriks pemeriksa paritas (*parity-check matrix) H* berukuran $r \times n$ dengan $r = n - k$. Baris-baris pada matriks ini membentuk basis dari ruang ortogonal terhadap C. Suatu vektor $x \in F_q^n$ merupakan *codeword* jika dan hanya jika memenuhi:

$$C = \{x \in F_q^n \mid H_x^T = 0\}.$$

Dengan kata lain, *C* merupakan himpunan solusi dari sistem persamaan linear homogen (SPLH) $Hx^T = 0$, atau disebut juga kernel dari *H*. Mengkonstruksi kode linear dengan panjang n dan berdimensi k sama artinya dengan mendefinisikan matriks cek paritas *H*.

Dalam proses *decoding*, matriks cek paritas berfungsi untuk menentukan simbol-simbol redundansi agar *codeword* yang terbentuk tetap berada di ruang solusi.

Misalkan vektor pesan sepanjang k:

$$u = (u_1, u_2, \dots, u_{\mathrm{k}}),$$

Yang kemudian di *encoding* menjadi *codeword* sepanjang n:

$$x = (x_1, x_2, \dots, x_n),$$

Dengan $x_1 = u_1, \dots, x_k = u_{\mathrm{k}}$. Sisa $r = (n - k)$ simbol berikutnya yaitu,

$$x_{k+1}, x_{k+2}, \dots, x_n,$$

adalah simbol cek (*parity symbols*) yang diperoleh dari penyelesaian SPL:

$$Hx^T = 0 \Leftrightarrow H\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Matriks *H* biasanya dinyatakan dalam bentuk standar, yaitu

$$H = [\, A \mid I_r \,], \tag{2.5}$$

di mana:

A : Matriks berukuran $r \times k$

I_r : Matriks identitas berukuran $r \times r$

Karena Reed–Solomon merupakan *evaluation code*, matriks *H* yang digunakan dalam penelitian ini dibentuk dengan cara mengevaluasi pangkat

elemen primitif α pada eksponen tertentu. Jika titik evaluasi untuk matriks generator menggunakan

$$x_j = \alpha^{5j},$$

maka titik evaluasi untuk matriks *parity-check* adalah titik dual, yaitu nilai-nilai $\alpha^{t(j)}$ untuk beberapa *t* derajat paritas. Baris-baris matriks *H* memiliki bentuk umum:

$$H_i = (1,\ \alpha^{5i},\ \alpha^{10i},\ \alpha^{15i},\ \alpha^{20i}, \dots,\ \alpha^{250i}),$$

di mana eksponen selalu dihitung modulo 255 (karena $\alpha^{255} = 1$ di $GF(2^8)$). Dengan demikian, matriks parity-check yang digunakan dapat dituliskan sebagai:

$$H = \begin{bmatrix} 1 & \alpha^5 & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \cdots & \alpha^{250} \\ 1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \cdots & \alpha^{245} \\ 1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \cdots & \alpha^{240} \\ 1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{160} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \cdots & \alpha^{235} \end{bmatrix}$$

3. *Codeword.*

Definisi 2.7

Diberikan $X = \{x_1, x_2, \dots, x_q\}$ adalah suatu himpunan yang berukuran q, yang dapat disebut alfabet kode dan elemen-elemennya disebut *codeword* (Ling & Xing, 2004).

a. Suatu word $q - er$ panjang n yaitu barisan $w = w_1, w_2, \dots, w_n$ dengan $w_i \in X$ untuk setiap i.
b. Kode blok $q - er$ dengan panjang n atas X merupakan himpunan tak kosong C pada word $q - er$ mempunyai panjang yang sama.
c. Elemen dari C disebut dengan *codeword*.
d. Kode dengan panjang n dan berukuran m disebut dengan kode-(n, m).

Contoh 2.4.

Suatu himpunan yang beranggotakan hewan yaitu $H = \{$kucing, anjing, burung, ikan, kelinci, ayam$\}$ akan dikodekan menjadi suatu pesan rahasia yang terdiri dari angka biner 0 dan 1 dengan panjang 3-bit pada Tabel 2.7.

Tabel 2.7 Tabel Korespondensi Hewan dan Kode Biner

Hewan	**Kode**
Kucing	000
Anjing	001
Burung	010
Ikan	011
Kelinci	100
Ayam	101

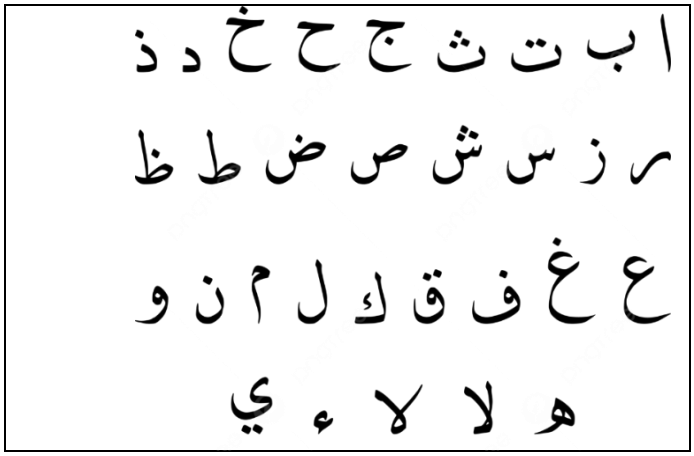
Jadi himpunan H dapat ditulis dalam bentuk kode yaitu,

$$H = \{000,001,010,011,100,101\}.$$

2.2.3 Pengkodean Huruf Hijaiyah

Huruf hijaiyah adalah abjad yang digunakan dalam penulisan teks Al-Qur'an. Kata huruf berasal dari bahasa Arab"*harf*" (حرف), dan bentuk jamaknya adalah "ḥurūf" (حروف). Huruf Arab disebut juga huruf hija'iyah (هجائية). Kata hija'iyah berasal dari kata kerja hajjaa (هجى) yang artinya mengeja, menghitung huruf, atau membaca huruf demi *huruf*. (Nasution, 2020).

Adapun huruf-huruf hijaiyah dapat dilihat pada Gambar 2.1.



Gambar 2.1 Huruf Hijaiyah

Pada penelitian ini, teks yang akan dikodekan menggunakan kode Reed-Solomon adalah ayat-ayat Al-Qur'an yang ditulis menggunakan huruf hijaiyah tanpa menggunakan harakat. Untuk mengubah ayat-ayat tersebut menjadi bentuk yang dapat diolah dalam sistem pengkodean, huruf-huruf hijaiyah dikonversikan menjadi angka-angka berdasarkan representasi *Unicode*, yaitu sistem pengkodean karakter universal yang memberikan setiap huruf Arab kode numerik unik dalam bentuk bilangan desimal dan biner 16-bit. Sehingga membentuk vektor dalam ruang vektor $V_k(F)$. Representasi ini menjadi dasar dalam proses *encoding* dan *decoding* kode Reed-Solomon untuk mendeteksi serta mengoreksi kesalahan penulisan ayat Al-Qur'an.

Tabel 2.8 menyajikan nilai *Unicode* dari huruf-huruf hijaiyah dasar dalam format Hex, Desimal, dan biner 16-bit. Nilai *Unicode* (Hex & Desimal), sedangkan kolom Representasi Biner (16-bit) merupakan hasil konversi peneliti (nilai *Unicode* desimal diubah menjadi biner 16-bit).

Tabel 2.8 Korespondensi Huruf Hijaiyah dan Vektor Biner

No	Huruf Hijaiyah	Nama Karakter	Unicode (Hex)	Unicode (Desimal)	Representasi Biner (16-bit)
1	ء	Hamzah	$U+0621$	1569	0000011000100001
2	أ	Alif Hamzah di atas	$U+0623$	1571	0000011000100011
3	ؤ	Wāw Hamzah	$U+0624$	1572	0000011000100100
4	إ	Alif Hamzah di bawah	$U+0625$	1573	0000011000100101
5	ئ	Yā' Hamzah	$U+0626$	1574	0000011000100110
6	ا	Alif	$U+0627$	1575	0000011000100111
7	ب	Bā'	$U+0628$	1576	0000011000101000
8	ت	Tā'	$U+062A$	1578	0000011000101010
9	ث	Thā'	$U+062B$	1579	0000011000101011
10	ج	Jīm	$U+062C$	1580	0000011000101100
11	ح	Ḥā'	$U+062D$	1581	0000011000101101
12	خ	Khā'	$U+062E$	1582	0000011000101110
13	د	Dāl	$U+062F$	1583	0000011000101111
14	ذ	Dhāl	$U+0630$	1584	0000011000110000
15	ر	Rā'	$U+0631$	1585	0000011000110001
16	ز	Zāy	$U+0632$	1586	0000011000110010
17	س	Sīn	$U+0633$	1587	0000011000110011
18	ش	Shīn	$U+0634$	1588	0000011000110100
19	ص	Ṣād	$U+0635$	1589	0000011000110101
20	ض	Ḍād	$U+0636$	1590	0000011000110110
21	ط	Ṭā'	$U+0637$	1591	0000011000110111
22	ظ	Ẓā'	$U+0638$	1592	0000011000111000
23	ع	ʿAyn	$U+0639$	1593	0000011000111001
24	غ	Ghayn	$U+063A$	1594	0000011000111010
25	ف	Fā'	$U+0641$	1601	0000011001000001
26	ق	Qāf	$U+0642$	1602	0000011001000010
27	ك	Kāf	$U+0643$	1603	0000011001000011
28	ل	Lām	$U+0644$	1604	0000011001000100
29	م	Mīm	$U+0645$	1605	0000011001000101
30	ن	Nūn	$U+0646$	1606	0000011001000110
31	ه	Hā'	$U+0647$	1607	0000011001000111
32	و	Wāw	$U+0648$	1608	0000011001001000
33	لا	Lam Alif	$U+FEFB$	65275	1111111011111011
34	ي	Yā'	$U+064A$	1610	0000011001001010
35	ى	Alif Maqṣūrah	$U+0649$	1609	0000011001001001
36	ة	Tā' Marbūṭah	$U+0629$	1577	0000011000101001
37	ـ	Spasi	$U+0020$	32	0000000000100000

2.2.4 Kode Reed-Solomon

Kode Reed-Solomon (RS) merupakan salah satu jenis kode blok linier yang bersifat siklik non-biner, didefinisikan atas lapangan hingga $GF(q)$, dan $GF(2^m)$. Kode ini pertama kali diperkenalkan oleh Irving S. Reed dan Gustave Solomon pada tahun 1960 dalam publikasi berjudul *"Polynomial Codes over Certain Finite Fields"*. Kode Reed-Solomon merupakan kode blok, yang berarti pesan yang akan ditransmisikan dibagi menjadi blok-blok data yang terpisah. Kode ini disebut juga kode sistematik yang artinya proses *encoding* tidak merubah simbol-simbol pesan dan simbol proteksi ditambahkan pada tempat yang terpisah pada blok data tersebut. Kode Reed-Solomon disebut juga kode linear (dengan menjumlahkan dua *codeword* akan menghasilkan *codeword* yang lain), dan juga siklik (dengan menggeser sebuah *codeword* secara siklik akan menghasilkan *codeword* lain). Reed-Solomon Code termasuk dalam keluarga pengkodean *Bose-Choundhuri-Hocquenghem* (BCH) non biner.

Kode Reed–Solomon merupakan salah satu jenis kode koreksi kesalahan yang bekerja dengan menambahkan simbol-simbol redundansi pada pesan asli sehingga sistem mampu mendeteksi dan memperbaiki kerusakan yang terjadi selama proses transmisi. Pada tahap *encoding*, pesan yang telah direpresentasikan dalam bentuk simbol pada lapangan hingga ditambahkan sejumlah simbol paritas yang berfungsi sebagai penanda untuk melacak kesalahan. Ketika *codeword* dikirim melalui kanal, sebagian simbol dapat berubah akibat gangguan. Pada saat penerimaan, *decoder* akan memeriksa konsistensi hubungan matematis antar-simbol untuk menentukan apakah terjadi kesalahan. Jika ditemukan ketidakcocokan, sistem kemudian mengidentifikasi posisi simbol yang rusak

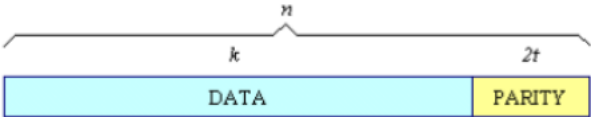
dengan memanfaatkan informasi yang tersimpan pada simbol-simbol paritas. Setelah posisi kesalahan diketahui, Reed–Solomon menghitung nilai asli simbol tersebut dan menggantinya dengan nilai yang benar sehingga *codeword* kembali valid.

Sebagai ilustrasi, misalkan surah *Al-Kahf* : 48 telah melalui proses *encoding* dan menghasilkan sebuah *codeword* yang berisi 51 simbol. Ketika dikirim, satu simbol berubah akibat gangguan. Pada tahap penerimaan, *decoder* mendeteksi bahwa susunan simbol tidak lagi sesuai dengan struktur matematis yang seharusnya. Sistem kemudian menelusuri posisi kesalahan tersebut misalnya pada simbol ke-31 dan menghitung nilai simbol yang benar berdasarkan informasi paritas. Setelah simbol itu diganti, *codeword* kembali sesuai dengan kondisi awal, dan surah *Al-Kahf* : 48 dapat dipulihkan sepenuhnya tanpa kehilangan satu huruf pun. Proses yang sama juga berlaku untuk dua, tiga, hingga lima kesalahan, selama jumlah kerusakan tidak melampaui batas kemampuan koreksi kode Reed–Solomon.

Dalam sistem komunikasi digital, kode Reed-Solomon memiliki dua proses:

1. *Encoding,* yaitu proses mengubah pesan menjadi *codeword* dengan menambahkan simbol paritas.
2. *Decoding,* yaitu proses mengembalikan *codeword* menjadi pesan asli sekaligus melakukan deteksi dan koreksi kesalahan yang mungkin terjadi selama transmisi.

Diagram struktur kode Reed-Solomon secara umum dapat dilihat pada Gambar 2.2.



Gambar 2.2 Diagram Struktur Kode Reed–Solomon $(n, k, 2t)$

Kode Reed-Solomon atas $GF(2^m)$ umumnya menggunakan parameter sebagai berikut:

1. $n = 2^m - 1$, panjang *codeword* dalam simbol. (2.6)
2. $k = n - 2t$, jumlah simbol data (pesan). (2.7)
3. $2t = n - k$, jumlah simbol paritas, dengan t merupakan jumlah simbol yang dapat dikoreksi sehingga $t = \left\lfloor \frac{n-k}{2} \right\rfloor$. (2.8)
4. $d_{min} = 2t + 1 = n - k + 1$, jarak minimum antar-*codeword*. (2.9)

Kode Reed-Solomon biasanya dilambangkan sebagai:

$$RS(n, k) \text{ atas } GF(q)$$

Definisi 2.8

Kode Reed-Solomon $RS(n, k)$ pengoreksi t-kesalahan adalah sebuah kode BCH primitif dengan panjang $n = q - 1$ atas lapangan $GF(q^m)$.

Contoh 2.5.

Misalkan lapangan hingga $GF(2^4)$ dan parameter:

$$n = 15, \quad k = 11$$

Maka:

$$t = \left\lfloor \frac{n-k}{2} \right\rfloor = \left\lfloor \frac{15-11}{2} \right\rfloor = 2$$

Sehingga, kode $RS(15, 11)$ mampu mendeteksi hingga 4 kesalahan dan mongerksi hingga 2 kesalahan dalam setiap *codeword*.

2.2.5 Proses *Encoding* dan *Decoding* Kode Reed-Solomon

Setelah konsep dasar kode siklik, kode linear, matriks generator, dan matriks parity check dipaparkan pada bagian sebelumnya, maka pada subbab ini dibahas proses pembentukan *codeword* menggunakan matriks generator khusus pada kode Reed–Solomon. Pembahasan difokuskan pada bagaimana vektor pesan diolah melalui operasi aritmetika pada $GF(2^8)$ sehingga menghasilkan simbol-simbol redundansi yang diperlukan untuk mendeteksi dan mengoreksi kesalahan pada tahap *decoding*.

1. Proses *Encoding.*

   Pada penelitian ini, proses *encoding* pada kode Reed-Solomon dilakukan dengan merepresentasikan pesan sebagai vektor yang beranggotakan elemen-elemen dari lapangan hingga $GF(2^m)$. Misalkan diberikan pesan,

   $$M = (M_0, M_1, \dots, M_{k-1})$$

   Dengan setiap $M_i \in GF(2^m)$, pesan tersebut kemudian dikodekan menjadi *codeword* berdimensi n melalui perkalian antara vektor pesan dengan matriks generator G, sehingga diperoleh:

   $$C = M \cdot G \tag{2.10}$$

   di mana:

   M : Vektor pesan berdimensi k, yaitu $(M_0, M_1, \dots, M_{k-1})$

   G : Matriks generator berukuran $k \times n$ yang dibentuk atas $GF(2^m)$

   Hasil dari *encoding* berupa codeword C berdimensi n, yang terdiri atas k simbol informasi dan $n - k$ simbol redundansi (paritas). Simbol redundansi inilah yang memungkinkan kode Reed–Solomon mendeteksi dan mengoreksi kesalahan hingga sebanyak t kesalahan.

2. Transmisi melalui *channel.*

   Dalam sistem komunikasi digital, transmisi merupakan jalur yang menghubungkan pengirim dengan penerima, yang bisa berupa kabel, udara, atau media lainnya. Setelah proses *encoding* selesai, data pesan yang telah dikodekan menjadi *codeword* $C(x)$ dikirimkan melalui kanal transmisi. Pada proses transmisi inilah kemungkinan terjadi gangguan (*noise*), interferensi atau kerusakan media, sehingga sebagian simbol dari codeword mungkin berubah. Untuk mensimulasikan kondisi tersebut, dalam implementasi program dilakukan penyisipan *error* (*error injection*) pada satu atau beberapa simbol *codeword*.

   Kesalahan yang terjadi direpresentasikan oleh polinomial *error* $e(x)$, yang menyatakan posisi dan besarnya kesalahan pada *codeword*. Dengan demikian, simbol yang diterima oleh penerima tidak lagi identik dengan simbol yang dikirim, melainkan berbentuk vektor penerimaan $v(x)$, yang secara matematis dimodelkan sebagai:

$$v(x) = C(x) + e(x) \tag{2.11}$$

   atau dalam bentuk vektor:

$$r = C + e \tag{2.12}$$

   di mana:

   C : $(C_0, C_1, \dots, C_{n-1})$ adalah *codeword* yang dikirim oleh pengirim, hasil dari proses *encoding*.

   r : $(r_0, r_1, \dots, r_{n-1})$ adalah *received word*, yaitu deretan simbol yang diterima penerima.

   e : $(e_0, e_1, \dots, e_{n-1})$ adalah vektor *error*.

Vektor *error* e menggambarkan posisi simbol yang rusak selama transmisi. Jika $e_i = 0$, maka simbol ke-i tidak mengalami kerusakan. Sebaliknya, jika $e_i \neq 0$, maka simbol pada posisi tersebut mengalami *error*. Dengan demikian:

a. Jika tidak ada *error* sama sekali, maka $r = c$.
b. Jika terdapat *error* pada beberapa posisi, maka $r \neq c$, dan selisihnya ditentukan oleh nilai e.

Contoh 2.6.

Misalkan kode Reed-Solomon $RS(7,4)$ atas $GF(2^3)$, dengan parameter $n = 7$, $k = 4$ dan $t = 2$. Pesan yang akan dikirim:

$$M = 1011$$

Hasil *encoding* (setelah penambahan simbol paritas) diperoleh *codeword*:

$$C = 1011010$$

Transmisi melalui channel dimodelkan dengan:

$$r = C + e$$

dengan e adalah vektor *error*.

Jika terjadi *error* pada bit ke-3, maka:

$$e = 001000$$

Sehingga:

$$r = 1011010 + 001000 = 1001010$$

3. Proses *Decoding*.

Proses *decoding* merupakan tahap untuk mengembalikan *codeword* yang diterima menjadi pesan asli. Pada tahap ini dilakukan dua langkah utama, yaitu proses deteksi kesalahan dan koreksi kesalahan. Jika *codeword* yang

diterima sama dengan yang dikirim, maka pesan dapat langsung dibaca tanpa koreksi. Namun, jika terdapat perbedaan akibat *noise* atau gangguan pada kanal, maka *decoder* harus mendeteksi lokasi kesalahan dan memperbaiki.

a. Perhitungan Sindrom.

Pada proses *decoding* Reed-Solomon, sindrom digunakan untuk mendeteksi keberadaan dan informasi tentang kesalahan dalam sebuah *codeword*. Jika vektor yang diterima adalah $v = (v_0, v_1, \dots, v_{n-1})$, maka sindrom ke-$j$ didefinisikan sebagai hasil evaluasi dari polinomial $v(x)$ pada akar-akar polynomial generator:

$$S_j = \sum_{i=0}^{n-1} v_i \cdot \alpha^{j \cdot i},\ j = 1,2, \dots ,2t \qquad (2.13)$$

dengan α adalah elemen primitif dari $GF(q)$, dan S_j adalah sindrom ke$-j$. Secara matriks, sindrom dapat dihitung dengan mengalikan matriks evaluasi H dengan transpose vektor (v^T) (Bras-amor, 2018), yaitu:

$$S = H \cdot v^T \qquad (2.14)$$

dengan *H* adalah matriks *parity-check* berukuran $2t \times n$:

$$H = \begin{bmatrix} 1 & \alpha^1 & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & \alpha^{2t \cdot 2} & \cdots & \alpha^{2t(n-1)} \end{bmatrix}$$

Apabila semua sindrom bernilai nol ($S_1 = S_2 = \cdots = S_{2t} = 0$), maka vektor penerimaan v adalah *codeword* yang valid. Sebaliknya, jika terdapat sindrom yang bernilai tidak nol, maka hal ini menunjukkan adanya *error* pada *codeword* yang diterima.

b. Menentukan Polinomial Lokasi kesalahan dan Posisi Kesalahan.

Dalam *decoding* Reed-Solomon, *error locator polynomial* $\sigma(x)$ digunakan untuk menentukan posisi simbol yang mengalami kesalahan. Polinomial ini didefinisikan sebagai:

$$\sigma(\mathrm{x}) = \prod_{j=1}^{v}(1 - \beta_j x) = 1 + \sigma_1 + \mathrm{x}\sigma_2\mathrm{x}^2 + + \cdots \sigma_v\mathrm{x}^v \qquad (2.15)$$

di mana:

β_j : α

v : Jumlah kesalahan

σ_i : Koefisien polinomial lokasi kesalahan

Akar-akar kebalikan dari $\sigma(x)$ menunjukkan posisi kesalahan pada *codeword*. Jika:

$$\sigma(\alpha^{-e}) = 0,$$

Jika $\alpha^{-e} = 0$ adalah akar dari $\sigma(x)$, maka posisi ke-e pada *codeword* terjadi kesalahan.

Untuk memperoleh $\sigma(x)$, digunakan *Berlekamp–Massey Algorithm* (BMA), yaitu algoritma yang mencari polinomial dengan derajat terkecil yang memenuhi relasi linier:

$$\sigma_0 S_j + \sigma_1 S_{j-1} + \sigma_2 S_{j-2} + \cdots + \sigma_L S_{j-L} = 0,$$

untuk $j = L + 1, \ldots, 2t,$

di mana:

S_j : Sindrom $ke - j$

σ_i : Koefisien polinomial *locator* kesalahan

L : Derajat polinomial *locator* (jumlah *error* yang ditemukan)

t : Jumlah *error* yang bisa dikoreksi

Dalam implementasi RS tertentu (misal $RS(255)$ dengan $s = 5$), posisi simbol ditentukan dalam langkah tertentu:

$$i = \frac{e}{s}$$

c. Mencari akar Polinomial Evaluasi Kesalahan (*Error*).

Setelah polinomial *locator* $\sigma(x)$ diperoleh, langkah berikutnya adalah menentukan akar polinomial evaluasi *error* $\Omega(x)$. Polinomial ini berfungsi untuk menghitung besar *error* pada setiap posisi kesalahan.

$$\Omega(x) = \mathrm{S}(x) \cdot \sigma(x)\ mod\ x^{2t}, \tag{2.16}$$

dengan,

$$S(x) = S_1 + S_2 x + \cdots + S_{2t} x^{2t-1},$$

merupakan polinomial sindrom.

d. Penentuan Nilai Magnitude Kesalahan (*Error Magnitude).*

Setelah posisi kesalahan dan akar polinomial evaluasi kesalahan ditemukan, langkah selanjutnya dalam proses *decoding* adalah menentukan besarnya nilai kesalahan yang terjadi pada setiap posisi tersebut. Untuk melakukan hal ini, digunakan Algoritma Forney yang menghitung *magnitudo error* pada setiap posisi kesalahan tanpa mengganggu simbol lainnya dalam vektor penerimaan. Besarnya nilai magnitude *error* pada posisi j dinotasikan dengan E_j, dan dapat dihitung dengan:

$$S_i = \sum_{j=1}^{v} e_j \cdot \beta_j^i, \quad i = 1,2, \ldots, 2t$$

dengan e_j adalah nilai kesalahan (*error value*) yang secara konseptual terjadi pada posisi β_j.

$$E_j = -\frac{\Omega(\mathrm{X_j^{-1}})}{\sigma\prime(\mathrm{X_j^{-1}})} \tag{2.17}$$

di mana:

$\Omega(x)$: Polinomial evaluasi *error* (*error evaluator polynomial*)

$\sigma(x)$: Polinomial *locator* kesalahan (*error locator polynomial*)

$\sigma'(x)$: Turunan formal dari $\sigma(x)$

$\mathrm{X_j^{-1}}$: Invers dari posisi *error* dalam bentuk eksponensial elemen primitif lapangan (α^{-j})

E_j : Besarnya nilai magnitude *error* yang terjadi di posisi ke-j

e. Koreksi Kode Kesalahan (*Error).*

Jika v adalah polinomial vektor penerimaan dan e adalah polinomial *error* (yang telah diketahui posisinya dan nilainya), maka *codeword* yang benar c dapat diperoleh dengan:

$$C = v - e$$

Karena pengurangan dalam lapangan hingga $GF(2^m)$ sama dengan penjumlahan (karena karakteristiknya 2), maka:

$$C = v \oplus e \tag{2.18}$$

di mana $\oplus$ menyatakan operasi XOR pada koefisien-koefisien dari vektor biner atau elemen-elemen $GF(2^m)$.

2.3 Transmisi

Teori kode pengoreksi *error* merupakan salah satu cabang matematika yang bergerak dibidang transmisi dan penyimpanan data. Media informasi tidak selalu memberikan keakuratan dalam menerima informasi, adakalanya terjadi suatu gangguan saat pengiriman pesan/informasi. Apabila terjadi suatu *error* pada saat pengiriman pesan/informasi, kesalahan tetap dapat terdeteksi bahkan diperbaiki dengan menambahkan suatu redundansi ke dalam pesan/informasi yang telah diubah dalam bentuk kode. Diagram sistem transmisi informasi secara umum dapat dilihat pada Gambar 2.3.

Gambar 2.3 Diagram Proses Pengiriman Pesan/Informasi

2.3.1 Transmisi Data

Transmisi data merupakan proses pengiriman informasi (data) yang telah dikonversi ke dalam bentuk kode tertentu melalui suatu media dari satu titik ke titik lainnya. Seiring berkembangnya teknologi, komunikasi data didefinisikan sebagai proses pengiriman dan penerimaan data atau informasi dari dua atau lebih perangkat (*device*) yang saling terhubung dalam sebuah jaringan, baik jaringan lokal (*Local Area Network*) maupun jaringan luas seperti internet.

Pada sistem transmisi data, terdapat media transmisi yang berfungsi sebagai jalur fisik penghubung antara pengirim dan penerima sinyal. Media ini dapat berupa media terpandu (*guided media*) seperti kabel tembaga, kabel koaksial, kabel *twisted pair*, dan serat optik, maupun media tidak terpandu (*unguided*

media) seperti gelombang radio, gelombang mikro, atau sinyal optik yang merambat melalui udara, ruang hampa, atau air.

Pada proses transmisi data sering menghadapi berbagai gangguan yang dapat menyebabkan kesalahan pada data yang dikirimkan. Kesalahan transmisi dapat berupa perubahan satu bit atau lebih dari data aslinya. Faktor-faktor yang dapat menyebabkan kesalahan dalam transmisi data:

1. Radiasi Elektromagnetik.

   Radiasi elektromagnetik merupakan gelombang energi yang dipancarkan oleh perangkat elektronik, seperti radio, *microwave*, atau perangkat komunikasi nirkabel lainnya. Radiasi ini dapat mengganggu sinyal yang sedang ditransmisikan melalui media komunikasi. Dampaknya, sinyal dapat mengalami distorsi, sehingga data yang diterima berbeda dari data yang dikirimkan

2. *Crosstalk* (sinyal bocor).

   Crosstalk terjadi ketika sinyal dari satu saluran komunikasi "bocor" ke saluran komunikasi lain yang berdekatan. Misalnya, pada kabel yang terhubung dengan perangkat yang saling berdekatan, atau dalam jaringan yang menggunakan saluran fisik yang sama. Ketika ini terjadi, sinyal yang seharusnya tetap utuh bisa terganggu oleh sinyal lain yang menyebabkan data menjadi rusak atau tidak akurat.

3. *Atenausi.*

   Atenuasi adalah pelemahan sinyal yang disebabkan oleh jarak antara penerima dan pengirim yang terlalu jauh. Ini dapat terjadi karena adanya

halangan diantara keduanya, misalkan sinyal WiFi yang semakin lemah ketika jauh dari *Router*.

4. *Noise* (gangguan sinyal).

   Noise merupakan masuknya sinyal lain yang tidak dibutuhkan oleh media dan menyebabkan sinyal menjadi hancur. Contoh dari *noise* adalah sinyal antenna televisi menjadi kabur karena adanya pesawat yang lewat.

5. Distorsi.

   Distorsi merupakan keadaan dimana sinyal yang dikirim berbeda dari media penerima sinyal sehingga membuat rusak media. Contoh distorsi adalah suara yang berisik pada speaker yang rusak.

Dampak dari gangguan-gangguan tersebut adalah munculnya kesalahan bit. Kesalahan bit terjadi ketika data yang diterima tidak sesuai dengan data yang dikirimkan. Bit adalah unit terkecil dari data digital yang hanya memiliki dua kemungkinan nilai, yaitu 0 atau 1. Ketika terjadi kesalahan bit, nilai "0" bisa berubah menjadi 1, atau sebaliknya, yang menyebabkan data yang diterima tidak sesuai dengan data yang seharusnya.

Adapun macam-macam bit *error* sebagai berikut:

1. *Single-bit error.*

   Single-bit error terjadi ketika hanya satu bit dalam unit data seperti bit, karakter, atau paket, berubah dari 1 ke 0 atau sebaliknya. Kesalahan jenis ini memiliki kemungkinan sangat kecil terjadi dalam transmisi data serial. Misalnya, jika data ditransmisikan pada kecepatan 1 Mbps, maka setiap bit hanya berlangsung selama 1 mikrodetik ($1\mu s$). Untuk menyebabkan *single bit error*, gangguan (*noise*) harus memiliki durasi yang sangat singkat, yaitu

sekitar $1\mu s$ yang jarang terjadi. Biasanya, *noise* berlangsung lebih lama dari itu, sehingga lebih mungkin menyebabkan kesalahan yang memengaruhi lebih dari satu bit.

2. *Burst error.*

   Burst error terjadi ketika terdapat dua atau lebih bit pada unit data telah berubah dari 1 ke 0 atau dari 0 ke 1. Pada kasus 0100010001000011 dikirim dan 0101110101100011 diterima, dapat diperhatikan bahwa beberapa bit mengalami perubahan tetapi tidak harus berurutan. Panjang *burst error* dihitung dari bit pertama yang rusak hingga bit terakhir yang rusak, meskipun ada bit diantaranya yang tetap benar.

   Burst error lebih sering terjadi dibandingkan *single-bit error* karena *noise* biasanya memiliki durasi lebih panjang dari 1 bit, yang berarti bahwa ketika *noise* mempengaruhi data, maka akan mempengaruhi satu set bit atau lebih dari satu bit. Jumlah bit yang terkena efeknya tergantung pada data *rate* dan durasi *noise* tersebut.

   Untuk mengatasi permasalahan gangguan dalam proses transmisi data, diperlukan suatu metode yang mampu mendeteksi serta memperbaiki kesalahan yang terjadi selama pengiriman. Tujuan utama dari metode ini adalah memastikan bahwa data yang diterima tetap akurat dan konsisten dengan data yang dikirim, meskipun terjadi gangguan pada saluran transmisi. Dengan demikian, informasi yang diterima oleh perangkat penerima dapat tetap dipercaya dan digunakan sebagaimana mestinya. Proses deteksi dan koreksi kesalahan bit berfungsi untuk mengenali serta memperbaiki kesalahan yang muncul pada bit-bit data selama proses

transmisi. Kedua proses ini merupakan komponen penting dalam sistem komunikasi digital, karena berperan besar dalam menjaga keandalan (*reliability*) dan integritas (*integrity*) data yang dikirimkan.

2.3.2 Deteksi dan Koreksi Kesalahan Bit Pada Transmisi Data

1. Deteksi Kesalahan Bit.

   Deteksi kesalahan bit merupakan proses untuk memastikan bahwa data yang diterima sesuai dengan data yang dikirim, tanpa adanya perubahan akibat gangguan selama transmisi. Teknik ini berperan penting dalam sistem komunikasi maupun penyimpanan data digital, karena proses transmisi sangat rentan terhadap gangguan seperti *noise*, interferensi atau kerusakan perangkat keras.

   Proses deteksi dilakukan dengan menambahkan bit kontrol atau bit redundansi pada data asli. Bit redudansi digunakan untuk memverifikasi kebenaran data saat diterima. Salah satu teknik yang umum digunakan adalah kode siklik (*cyclic code*), yang bekerja berdasarkan polinomial generator sebagai kunci pembentukan kode kesalahan.

   Dalam proses pengkodean, polinomial data dibagi dengan polinomial generator untuk memperoleh sisa pembagian (*remainder*), yang kemudian ditambahkan ke dalam data sebagai bagian dari kode kesalahan. Pada sisi penerima, pembagian serupa dilakukan menggunakan polinomial generator yang sama. Jika hasil pembagian menghasilkan sisa nol, maka data dianggap benar. Sebaliknya, jika sisa tidak nol, berarti telah terjadi kesalahan selama transmisi.

Kode siklik memiliki kemampuan deteksi yang lebih handal dibandingkan metode sederhana seperti bit paritas, karena dapat dirancang untuk mendeteksi berbagai pola kesalahan, termasuk kesalahan tunggal, kesalahan ganda, maupun *burst error*. Metode ini bersifat efisien, cepat, dan banyak diterapkan dalam berbagai sistem modern seperti komunikasi jaringan, penyimpanan digital, serta protokol komunikasi seperti *ethernet*.

2. Koreksi Kesalahan Bit.

   Koreksi kesalahan bit merupakan proses lanjutan setelah deteksi, yang tidak hanya mengidentifikasi adanya kesalahan dalam data, tetapi juga memperbaikinya secara otomatis tanpa perlu pengiriman ulang. Teknik ini sangat penting dalam sistem komunikasi dan penyimpanan data digital untuk menjaga keandalan transmisi serta integritas informasi.

   Teknik koreksi kesalahan bekerja dengan menambahkan informasi tambahan atau kode khusus ke dalam data asli saat proses pengiriman. Infomrasi ini dirancang sedemikian rupa sehingga memungkinkan sistem penerima untuk memverifikasi integritas data yang diterima. Apabila terjadi kesalahan, sistem dapat menentukan lokasi bit yang mengalami perubahan dan mengembalikannya ke nilai yang seharusnya.

   Langkah penting dalam koreksi adalah perhitungan nilai sindrom (*syndrome calculation*), yaitu hasil operasi pembagian polinomial data yang diterima dengan polinomial generator. Nilai sindrom menunjukkan apakah terjadi kesalahan, sekaligus membantu menentukan lokasi kesalahan. Jika nilai sindrom sama dengan nol, maka data dianggap bebas kesalahan. Namun, jika tidak nol, sistem akan menggunakan algoritma *decoding* seperti *Berlekamp-Massey* atau *Chien Search* untuk menemukan posisi kesalahan.

Setelah posisi kesalahan diketahui, sistem melakukan pembalikan nilai bit yang rusak sehingga pesan asli dapat direkonstruksi dengan benar. Proses koreksi kesalahan ini memberikan lapisan perlindungan tambahan terhadap kerusakan data, terutama pada sistem di mana transmisi ulang tidak dimungkinkan atau memerlukan waktu serta sumber daya yang besar.

2.3.3 Transmisi Ayat Al-Qur'an

Dalam penelitian mengenai "*Implementasi Kode Reed–Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur'an"*, proses transmisi didefinisikan sebagai proses pengiriman data hasil pengkodean huruf hijaiyah melalui saluran digital. Teks ayat Al-Qur'an terlebih dahulu dikonversi menjadi simbol pada lapangan hingga $GF(2^8)$, kemudian dikodekan menggunakan kode Reed–Solomon untuk mendapatkan *codeword*.

Pada proses transmisi, selalu terdapat kemungkinan terjadinya *error*. Untuk mensimulasikan gangguan seperti pada transmisi nyata, penelitian ini menyisipkan *error* secara sengaja (*injected error*) pada beberapa simbol dalam *codeword*. Penyisipan ini didasarkan pada pola kesalahan penulisan ayat Al-Qur'an di dunia nyata, misalnya pertukaran huruf (ب menjadi ت), huruf yang hilang, kesalahan titik, penambahan atau pengurangan karakter.

Kesalahan linguistik nyata ini hanya menjadi landasan dasar, namun dalam penelitian diterapkan sebagai kesalahan simbol pada proses transmisi, yaitu:

1. Kesalahan substitusi simbol.

   Satu simbol hasil *encoding* diganti dengan simbol lain (α^3 menjadi α^{11}). Ini menggambarkan *single-symbol error*.

2. Kesalahan *burst.*

   Dua atau lebih simbol diganti atau diubah sekaligus dalam satu *codeword*. Ini memodelkan kondisi *noise* panjang atau interferensi.

3. Kesalahan acak (*random error*).

   Penelitian dapat menyisipkan *error* secara acak di beberapa posisi untuk meniru kondisi saluran yang tidak stabil.

2.3.4 Proses Deteksi dan Koreksi Kesalahan pada Transmisi Ayat Al-Qur'an

Untuk menjaga keakuratan teks ayat Al-Qur'an yang dikirim, digunakan kode Reed–Solomon (RS). Reed–Solomon merupakan salah satu jenis *error-correcting code* yang mampu mendeteksi dan memperbaiki kesalahan pada blok simbol.

1. Deteksi Kesalahan (*Error Detection*).

   Setelah penerima mendapatkan *codeword*, langkah pertama adalah menghitung sindrom:

   $$S_i = v(\alpha^i), \quad i = 1,2,\dots,2t$$

   Sindrom bernilai nol ($S_1 = S_2 = \cdots = 0$) berarti:

   a. Tidak ada kesalahan, dan

   b. Codeword diterima dengan benar.

   Jika salah satu sindrom $\neq 0 \rightarrow$ berarti terjadi kesalahan.

2. Penentuan Lokasi Kesalahan (*Error Locator Polynomial*).

   Jika terjadi kesalahan, Reed–Solomon membentuk *error locator polynomial*:

   $$\Lambda(x) = 1 + \lambda_1 x + \cdots + \lambda_t x^t$$

Polinomial ini diperoleh melalui:

a. algoritma Berlekamp–Massey, atau

b. *Euclidean Algorithm*.

Akar-akar dari $\Lambda(x)$ menunjukkan posisi simbol yang salah.

3. Perhitungan Nilai Besar Kesalahan (*Error Magnitude*).

   Setelah menemukan posisi *error*, langkah selanjutnya adalah menghitung besar kesalahan menggunakan Rumus Forney:

$$E_j = -\frac{\Omega(Xj-1)}{\Lambda'(Xj-1)}$$

   Nilai ini menentukan seberapa besar simbol tersebut harus dikoreksi.

4. Koreksi Kesalahan (*Error Correction*).

   Simbol yang salah kemudian diperbaiki:

$$C_i = v_i - e_i$$

   Hasilnya adalah *codeword* yang telah kembali benar.

Pengiriman ayat Al-Qur'an harus memiliki tingkat akurasi tinggi (*zero-tolerance error*) karena:

1) Perubahan satu huruf dapat mengubah makna ayat,
2) Teks Al-Qur'an memiliki standar rasm yang baku,
3) Kesalahan penulisan dapat berpotensi menjadi kesalahan pemaknaan.

Dengan menyisipkan *error* secara terkendali pada proses transmisi, penelitian ini menunjukkan bahwa:

1) Model kesalahan dapat dideteksi,
2) Kesalahan dapat diperbaiki secara matematis,
3) Ayat Al-Qur'an dapat dipulihkan sesuai teks aslinya dengan kode Reed–Solomon.

2.4 Kajian Integrasi Topik Penelitian dengan Al-Qur’an

Mengintregasikan ilmu pengetahuan dengan ajaran Al-Qur’an dan Hadits merupakan pendekatan komprehensif yang memadukan antara perkembangan sains modern dan nilai-nilai keislaman. Pendekatan ini tidak hanya memperluas pemahaman terhadap aspek teknologi dan sains, tetapi juga memperkuat keyakinan bahwa menuntut ilmu dan mengaplikasikannya merupakan bagian penting dari keimanan dan penghambaan kepada Allah SWT. Penelitian yang berjudul "Implementasi Kode Reed-Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur’an Menggunakan Pengkodean Huruf Hijaiyah" merupakan salah satu bentuk nyata dari penggunaan teknologi informasi untuk menjaga keaslian dan ketepatan teks ayat Al-Qur’an. Dalam hal ini, penerapan metode koreksi kesalahan digital (*error correction*) tidak hanya berfungsi secara teknis, tetapi juga memiliki makna spiritual dalam mendukung upaya pelestarian kitab suci yang telah dijamin kemurniannya oleh Allah SWT. Berikut beberapa ayat Al-Quran dan Hadits yang relevan untuk mendukung dan memberikan perspektif religius pada penelitian ini.

1. Kewajiban Bersikap Teliti dan Tidak Mengikuti Sesuatu Tanpa Ilmu.

   Dalam ajaran Islam, sikap teliti, cermat, dan tidak tergesa-gesa dalam menerima ataupun menyebarkan informasi merupakan prinsip yang sangat ditekankan. Kehati-hatian tersebut menjadi landasan penting agar seseorang tidak terjerumus pada kesalahan atau kekeliruan dalam memahami maupun menyampaikan suatu perkara, sebagaimana disebutkan dalam Surah Al-Isra’ ayat 36 (Kementerian Agama, 2022):

وَلَا تَقْفُ مَا لَيْسَ لَكَ بِهِۦ عِلْمٌ ۚ إِنَّ السَّمْعَ وَالْبَصَرَ وَالْفُؤَادَ كُلُّ أُوْلَٰٓئِكَ كَانَ عَنْهُ مَسْـُٔولًا

"Dan janganlah kamu mengikuti sesuatu yang tidak kamu ketahui. Sesungguhnya pendengaran, penglihatan, dan hati, semuanya itu akan dimintai pertanggungjawaban."(QS. Al-Isra': 36)

Ayat ini merupakan prinsip dasar dalam Islam terkait keharusan untuk berhati-hati dalam menerima, menyampaikan, dan memproses informasi. Penulisan ayat Al-Qur'an yang salah baik satu huruf, harakat, maupun tanda baca dapat menyebabkan perubahan makna atau bahkan kesalahan fatal dalam bacaan. Oleh karena itu, deteksi dan koreksi penulisan ayat menjadi bentuk konkret dari perintah untuk tidak mengikuti sesuatu yang belum pasti (tidak valid). Penerapan metode ilmiah seperti kode Reed-Solomon, yang memiliki kemampuan dalam mendeteksi dan mengoreksi kesalahan berdasarkan prinsip matematika dan teori informasi, menunjukkan bagaimana ilmu pengetahuan dapat menjadi alat bantu dalam menjalankan amanah keagamaan, memastikan teks wahyu tetap utuh dan bebas dari kesalahan.

Ayat ini mengingatkan bahwa setiap informasi yang kita terima (melalui pendengaran, penglihatan, atau hati) harus diteliti dan dikonfirmasi kebenarannya, karena semuanya akan dimintai pertanggungjawaban. Dalam konteks teknologi digital, ketika teks Al-Qur'an didistribusikan melalui aplikasi, *e-book*, atau media daring, tanggung jawab ini semakin besar. Maka penggunaan teknologi koreksi otomatis adalah bagian dari ikhtiar amanah ilmiah dan spiritual.

2. Tanggung Jawab Menyampaikan Al-Qur’an dengan Benar.

خَيْرُكُمْ مَنْ تَعَلَّمَ الْقُرْآنَ وَعَلَّمَهُ

“Sebaik-baik kalian adalah orang yang belajar Al-Qur’an dan mengajarkannya.” (HR. Bukhari No. 5027)

Hadits ini menegaskan bahwa keutamaan umat Islam terletak pada aktivitas pembelajaran dan pengajaran Al-Qur’an. Namun, dalam era digital, bentuk belajar dan mengajar Al-Qur’an telah berkembang tidak hanya secara lisan dan tulisan manual, tetapi juga dalam bentuk digitalisasi teks dan audio Al-Qur’an yang tersebar di berbagai platform. Dalam proses digitalisasi, kesalahan penulisan atau konversi teks Arab sangat mungkin terjadi, apalagi jika prosesnya dilakukan tanpa deteksi atau sistem koreksi yang tepat. Hal ini sejalan dengan hadits di atas, belajar dan mengajarkan Al-Qur’an juga mencakup usaha melestarikan dan menyampaikan Al-Qur’an secara benar dan akurat, baik melalui media konvensional maupun teknologi modern.

BAB III
METODE PENELITIAN

3.1 Jenis Penelitian

Penelitian ini merupakan penelitian kuantitatif yang menggunakan pendekatan simulasi algoritmik. Pendekatan ini dipilih karena penelitian berfokus pada proses pengukuran dan analisis numerik terhadap performa sistem dalam mendeteksi serta mengoreksi kesalahan transmisi pada ayat-ayat Al-Qur'an yang dikodekan menggunakan huruf hijaiyah. Proses simulasi dilakukan melalui dua tahapan utama, yaitu *encoding* dan *decoding*, dengan menerapkan kode Reed-Solomon pada representasi digital teks Al-Qur'an yang telah dikonversi ke dalam format *Unicode* 16-bit.

Untuk menguji efektivitas sistem, dilakukan penyisipan kesalahan secara sistematis pada data teks ayat, kemudian diukur sejauh mana algoritma Reed-Solomon mampu mendeteksi dan mengoreksi kesalahan tersebut. Evaluasi dilakukan berdasarkan jumlah kesalahan yang berhasil dideteksi dan diperbaiki, serta tingkat akurasi pemulihan data terhadap teks hijaiyah asli. Selain itu, dilakukan pula simulasi manual sebagai langkah verifikasi guna memastikan kesesuaian antara hasil perhitungan teoritis dan hasil pengujian menggunakan perangkat lunak. Seluruh proses implementasi dan pengujian dalam penelitian ini dilakukan menggunakan perangkat lunak SageMath, yang memiliki kemampuan dalam melakukan operasi aritmetika pada lapangan hingga (*Galois Field*) serta mendukung mekanisme koreksi kesalahan otomatis pada kode Reed-Solomon.

3.2 Data dan Sumber Data

Data yang digunakan dalam penelitian ini berupa teks ayat-ayat Al-Qur'an yang direpresentasikan dalam bentuk biner (kode digital) menggunakan huruf-huruf hijaiyah berdasarkan sistem *Unicode* 16-bit, khususnya dalam rentang *Unicode Block Arabic* ($UTF-8$). Representasi ini memungkinkan setiap huruf hijaiyah dikodekan sebagai simbol digital yang dapat diproses secara matematis menggunakan metode pengkodean Reed-Solomon.

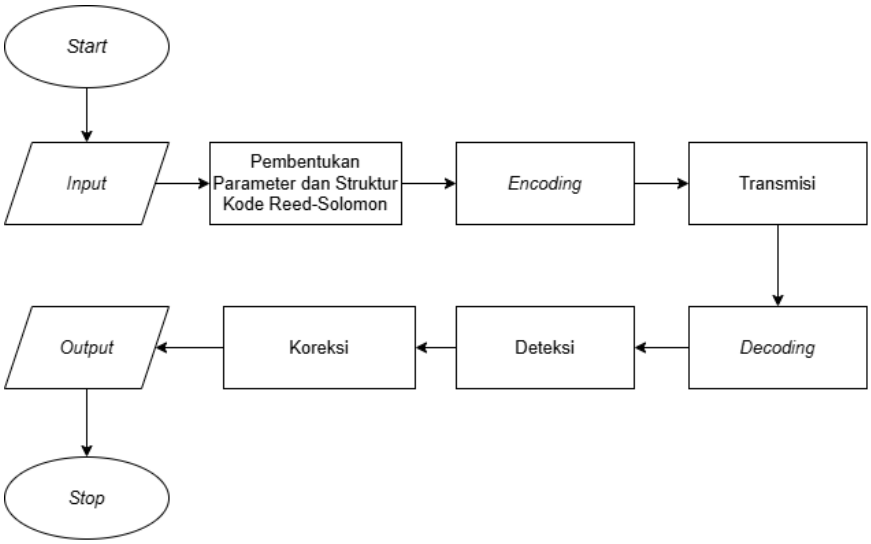
Meskipun penelitian mengacu pada kasus-kasus kesalahan penulisan huruf hijaiyah yang dapat ditemukan pada mushaf digital, aplikasi Al-Qur'an, atau situs web yang menampilkan teks berdasarkan Rasm Utsmani, data yang digunakan dalam simulasi tetap berupa teks ayat Al-Qur'an yang benar dengan simulasi disisipkan *error* secara terkontrol (*injected errors*) saat proses simulasi transmisi digital. Dengan demikian, kesalahan yang diuji bukan berasal dari kanal komunikasi fisik, tetapi merupakan model kesalahan sintetis yang dirancang untuk menguji kemampuan deteksi dan koreksi dari kode Reed-Solomon.

Setiap ayat diuji dengan variasi jumlah kesalahan sebanyak 1, 2, 3, 4, dan 5 kesalahan per ayat, sehingga menghasilkan berbagai kondisi kerusakan yang dapat dianalisis secara kuantitatif. Seluruh proses *encoding*, penyisipan kesalahan, perhitungan sindrom, deteksi letak kesalahan, dan proses koreksi dilakukan menggunakan perangkat lunak SageMath.

Pendekatan ini memungkinkan penelitian mensimulasikan kondisi kesalahan penulisan huruf hijaiyah sebagaimana terjadi dalam praktik digital, namun tetap memelihara kemurnian teks ayat suci Al-Qur'an. Dengan demikian, sistem dapat diuji secara realistis, terukur, dan tetap menjaga keaslian teks ayat Al-Qur'an.

3.3 Tahapan Penelitian

Berikut alur dari implementasi kode Reed-Solomon untuk mendeteksi dan mengoreksi kesalahan trasmisi ayat Al-Qur'an menggunakan pengkodean huruf hijaiyah:



Gambar 3.1 Alur Penelitian

1. *Input*

   Input penelitian ini meliputi data teks sepuluh ayat Al-Qur'an yang dikodekan dalam format *Unicode* 16 −bit sebagai dasar pembentukan simbol pada lapangan hingga $GF(2^8)$, serta parameter teknis kode Reed–Solomon yang mencakup panjang *codeword* $n = 51$, panjang pesan $k = 47$, nilai $m = 8$, dan variasi kemampuan koreksi t antara 1 sampai 5 kesalahan untuk setiap ayat. Selain itu, penelitian ini juga menggunakan *input* berupa pola kesalahan yang disisipkan secara terkontrol ke dalam *codeword*, yang mencakup jumlah kesalahan, posisi simbol yang mengalami gangguan, dan nilai magnitudo *error*, sehingga dapat dilakukan pengujian terhadap kemampuan deteksi dan koreksi kesalahan dari sistem Reed–Solomon.

2. Simulasi Pembentukan Parameter dan Struktur Kode Reed-Solomon (*RS*)

   Pada tahap awal dilakukan penetapan parameter-parameter dasar kode Reed-Solomon, yang meliputi panjang *codeword* (n) atau total simbol setelah *encoding*, panjang pesan (k), kemampuan koreksi kesalahan (t), serta pembentukan lapangan hingga $GF(2^m)$. Selanjutnya dilakukan pembentukan matriks generator (G) dan matriks *parity-check* (H) yang digunakan dalam proses *encoding* dan *decoding*.

   Langkah-langkah untuk simulasi pembentukan parameter dan struktur kode Reed-Solomon adalah sebagai berikut:

   a. Menetapkan Jumlah Bit per Simbol.

      Digunakan nilai $m = 8$, sehingga setiap simbol direpresentasikan dengan 8 bit dan operasi dilakukan dalam $GF(2^8)$.

   b. Menentukan Panjang *Codeword*.

      Panjang *codeword* n ditentukan menggunakan *Shortened Reed-Solomon code* sehingga $n = 51$.

   c. Menentukan Toleransi Kesalahan.

      Pada penelitian ini, perhitungan manual dilakukan dengan $t = 2$ kesalahan, sedangkan pada implementasi komputasi (*coding*) dilakukan pengujian dengan lima variasi jumlah kesalahan, yaitu $1,2,3,4,$ dan 5 kesalahan pada tiap ayat untuk menguji batas efisiensi sistem.

   d. Menghitung Panjang Pesan.

      Panjang pesan $k = 47$ diperoleh dari Persamaan (2.7). Sehingga parameter kode Reed-Solomon $RS(51, 47)$.

e. Membangkitkan Matriks Generator G.

Matriks generator G berukuran $k \times n$, dibentuk menggunakan perangkat lunak SageMath, berdasarkan evaluasi polinomial pesan pada titik-titik α^{ij} di $GF(2^m)$.

f. Membangkitkan Matriks *Parity-check* H.

Matriks *parity-check* H berukuran $(n-k) \times n$, di mana setiap baris merupakan evaluasi dari $\alpha^{j \cdot i}$ dengan $j = 1,2, \dots, n-k$ dan $i = 0,1, \dots, n-1$.

3. Simulasi *Encoding* Kode Reed-Solomon pada Kesalahan Penulisan Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah.

Proses *encoding* dalam kode Reed-Solomon dilakukan melalui beberapa tahapan sistematis. Adapun langkah-langkah dalam proses *encoding* sebagai berikut:

a. Menentukan Pesan Asli (*plaintext*).

Pesan asli direpresentasikan sebagai vektor pesan *M* dengan panjang *k*. Dalam penelitian ini, perhitungan manual menggunakan surah *Al-Kahfi* ayat 8:

"وانا لجعلون ما عليها صعيدا جرزا"

Sedangkan pada implementasi komputasi (*coding*) dan evaluasi performa, dilakukan pengujian terhadap $1-5$ kesalahan simbol kesalahan pada 10 ayat Al-Qur'an.

b. Mengonversi Teks ke dalam Format Biner *Unicode* 16-*bit*.

Setiap huruf Hijaiyah pada pesan dikonversi ke dalam representasi biner menggunakan format *Unicode* 16-bit berdasarkan *Unicode Block Arab* UTF-8.

c. Mengelompokkan Hasil Biner Menjadi Blok 8-*bit* dalam $GF(2^8)$.

Setiap $16-$ bit *Unicode* dipecah menjadi 2 simbol masing-masing $8-bit$, kemudian setiap blok direpresentasikan sebagai elemen pada lapangan hingga $GF(2^8)$. Representasi elemen pada lapangan menggunakan simbol α, yang merupakan akar primitif dari polinomial primitif $p(x)$.

d. Melakukan *Padding.*

e. Jika jumlah simbol pada blok pesan kurang dari panjang pesan k, maka dilakukan proses *padding* dengan menambahkan simbol "0" hingga panjang blok mencapai k. Proses ini bertujuan untuk menyeragamkan panjang setiap vektor pesan sehingga dapat diproses lebih lanjut pada tahap *encoding* tanpa memengaruhi informasi asli yang terkandung dalam pesan.

f. Melakukan Proses *Encoding*.

Proses *encoding* pada kode Reed-Solomon bertujuan untuk mengubah pesan asli menjadi sebuah *codeword* yang memiliki kemampuan deteksi dan koreksi. Dalam penelitian ini proses *encoding* dilakukan dengan mengalikan pesan (M) dengan matriks generator (G) untuk menghasilkan *codeword* (C) pada lapangan hinggan $GF(2^8)$ menggunakan Persamaan 2.10. Hasil akhir

dari proses ini adalah sebuah C berukuran n yang akan mengalami proses transmisi.

4. *Codeword*

   Codeword (C) adalah representasi dari data hasil proses *encoding* yang mengandung informasi asli beserta bit-bit redundansi. Jika terjadi gangguan atau kesalahan selama transmisi atau penyimpanan, *codeword* yang diterima (disebut vektor diterima) berbeda dari *codeword* asli. Vektor ini kemudian digunakan untuk proses koreksi kesalahan.

5. *Noise*

   Menambahkan *error* (e) sebanyak t secara acak pada *codeword* hasil *encoding*. Kesalahan ini direpresentasikan dalam bentuk polinomial *error* $e(x)$, sehingga membentuk vektor hasil penjumlahan antara *codeword* dan *error* $v(x) = C(x) + e(x)$.

6. Simulasi *Decoding* Kode Reed-Solomon pada Kesalahan Penulisan Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah.

   Setelah proses *encoding* dan penambahan kesalahan dilakukan, tahap selanjutnya adalah proses *decoding* untuk mendeteksi dan mengoreksi kesalahan yang terjadi pada data, sehingga informasi yang diterima dapat dikembalikan mendekati atau sama dengan pesan aslinya sesuai dengan kemampuan koreksi kode yang digunakan. Langkah-langkah proses *decoding* kode Reed-Solomon adalah sebagai berikut:

   a. Menghitung Sindrom.

      Sindrom dihitung dengan mengalikan matriks *parity-check* H dengan vektor *transpose* dari *codeword yang* diterima v^T, sebagaimana pada

Persamaan (2.14). Tahap perhitungan sindrom ini juga berfungsi sebagai proses deteksi kesalahan pada data yang diterima.

b. Menentukan Polinomial Lokasi Kesalahan (*Error Locator Polynomial*)

Polinomial locator kesalahan $\sigma(x)$ dan posisi kesalahan ditentukan menggunakan algoritma *Berlekamp–Massey* menggunakan Persamaan (2.15).

c. Mencari Akar Polinomial Evaluasi Kesalahan (*error*).

Menghitung akar polinomial evaluasi *error* $\Omega(x)$ menggunakan algoritma *Chien Search*, dengan Persamaan (2.16).

d. Menghitung Nilai Magnitude Kesalahan (*error magnitude*).

Menghitung nilai besar kesalahan E_j pada posisi j menggunakan algoritma *Forney*, dengan Persamaan (2.17).

e. Melakukan Proses *Decoding* untuk Memperoleh Kembali ke dalam pesan asli.

Setelah *codeword* diterima dan terdapat kemungkinan kesalahan, langkah *decoding* dilakukan untuk mengoreksi simbol yang salah. Dengan menggunakan sindrom, polinomial *locator*, dan algoritma Forney, diperoleh posisi dan magnitudo *error*. Setiap simbol yang salah kemudian diperbaiki sehingga diperoleh *codeword* terkoreksi (C) dalam bentuk simbol $GF(2^m)$:

$$C = v - e$$

Hasil decoding ini tetap berada dalam domain $GF(2^m)$ dan belum dikonversi menjadi biner maupun teks, sehingga informasi asli terjaga sepenuhnya.

7. *Output*

Output pada penelitian ini berupa teks ayat Al-Qur'an yang telah dikoreksi otomatis, tingkat keberhasilan deteksi kesalahan, dan tingkat keberhasilan koreksi kesalahan. Evaluasi dilakukan berdasarkan:

a. Jumlah kesalahan yang terdeteksi dengan benar.
b. Jumlah kesalahan berhasil dikoreksi.
c. Kondisi sistem masih gagal mengoreksi (*over* t).

BAB IV
HASIL DAN PEMBAHASAN

4.1 Simulasi Pembentukan Parameter dan Struktur Kode Reed-Solomon

Dalam kode Reed-Solomon, parameter n menyatakan panjang total dari blok kode (*codeword*), yaitu jumlah simbol setelah proses *encoding*. Sementara itu, parameter k merupakan jumlah simbol pesan asli yang dikodekan sebelum ditambahkan simbol redudansi untuk keperluan deteksi dan koreksi kesalahan. Dengan demikian, $n - k$ menyatakan banyaknya simbol *parity* yang ditambahkan pada proses *encoding* untuk meningkatkan kemampuan koreksi kesalahan.

Langkah pertama dalam menentukan nilai n dan k adalah menetapkan nilai m, yang menunjukkan banyaknya bit dalam satu simbol atau derajat dari lapangan hingga $GF(2^m)$. Nilai m harus berupa bilangan bulat positif karena menentukan ukuran dan elemen dari lapangan hingga yang digunakan. Dalam penelitian ini digunakan $m = 8$, sehingga operasi dilakukan pada lapangan hingga $GF(2^8)$. Berdasarkan teori kode Reed-Solomon, panjang maksimum blok kode ditentukan oleh persamaan $n = 2^m - 1 = 2^8 - 1 = 255$, sehingga untuk $m = 8$ diperoleh panjang maksimum $n = 255$.

Namun, karena ukuran data yang dikodekan dalam penelitian ini tidak mencapai 255 simbol, maka digunakan bentuk *shortened Reed-Solomon code* atau kode RS yang dipendekkan. *Shortened code* merupakan versi kode Reed-Solomon yang panjangnya dikurangi dari panjang maksimum dengan cara menghilangkan sejumlah simbol awal dari pesan. Proses pemendekan ini tidak mengubah karakteristik dasar maupun kemampuan koreksi kesalahan dari kode RS asalnya,

tetapi menyesuaikan panjang *codeword* agar sesuai dengan ukuran data yang sebenarnya digunakan.

Dalam penelitian ini digunakan kode $RS(51, k)$, yang merupakan hasil pemendekan dari kode maksimum $RS(255, 251)$ di lapangan hingga $GF(2^8)$. Artinya, setiap blok *codeword* terdiri atas 60 simbol pesan asli dan 4 simbol *parity*, dengan kemampuan untuk mengoreksi hingga dua simbol kesalahan dalam setiap blok. Penggunaan kode RS yang dipendekkan ini dipilih karena ukuran data (ayat penelitian) hanya memiliki total 512 bit atau setara dengan 64 simbol di $GF(2^8)$, sehingga parameter $n = 51$ menjadi pilihan yang efisien dan sesuai dengan kebutuhan sistem.

$n = 51$

$t = 2$

$k = n - 2t = 51 - 2 \cdot 2 = 47$

Pada penelitian ini, perhitungan manual dilakukan dengan mensimulasikan dua kesalahan (*error*) untuk menguji kemampuan sistem dalam melakukan koreksi hingga $t = 2$ kesalahan. Dengan demikian, parameter yang digunakan adalah $n = 51$, $k = 47$, dan $t = 2$. Lapangan hingga $GF(2^8)$ digunakan dengan representasi simbol berdasarkan akar primitif α yang telah dicantumkan pada Tabel 2.5.

Matriks generator G dibentuk menggunakan perangkat lunak *sagemath* berdasarkan parameter kode yang telah ditentukan, yaitu $n = 51$ dan $k = 47$. Dengan demikian, diperoleh matriks generator G berukuran 47×51 sebagai berikut:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha^{5} & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \alpha^{65} & \alpha^{70} & \alpha^{75} & \alpha^{80} & \cdots & \alpha^{250} \\ 1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \alpha^{130} & \alpha^{140} & \alpha^{150} & \alpha^{160} & \cdots & \alpha^{245} \\ 1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \alpha^{195} & \alpha^{210} & \alpha^{222} & \alpha^{240} & \cdots & \alpha^{240} \\ 1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{140} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \alpha^{5} & \alpha^{25} & \alpha^{45} & \alpha^{65} & \cdots & \alpha^{235} \\ 1 & \alpha^{25} & \alpha^{50} & \alpha^{75} & \alpha^{100} & \alpha^{125} & \alpha^{150} & \alpha^{175} & \alpha^{200} & \alpha^{225} & \alpha^{250} & \alpha^{20} & \alpha^{45} & \alpha^{70} & \alpha^{95} & \alpha^{120} & \alpha^{145} & \cdots & \alpha^{230} \\ 1 & \alpha^{30} & \alpha^{60} & \alpha^{90} & \alpha^{120} & \alpha^{150} & \alpha^{180} & \alpha^{210} & \alpha^{240} & \alpha^{15} & \alpha^{45} & \alpha^{75} & \alpha^{105} & \alpha^{135} & \alpha^{165} & \alpha^{195} & \alpha^{225} & \cdots & \alpha^{225} \\ 1 & \alpha^{35} & \alpha^{70} & \alpha^{105} & \alpha^{140} & \alpha^{175} & \alpha^{210} & \alpha^{245} & \alpha^{25} & \alpha^{60} & \alpha^{95} & \alpha^{130} & \alpha^{165} & \alpha^{200} & \alpha^{235} & \alpha^{15} & \alpha^{50} & \cdots & \alpha^{220} \\ 1 & \alpha^{40} & \alpha^{80} & \alpha^{120} & \alpha^{160} & \alpha^{200} & \alpha^{240} & \alpha^{25} & \alpha^{65} & \alpha^{105} & \alpha^{9} & \alpha^{185} & \alpha^{225} & \alpha^{10} & \alpha^{50} & \alpha^{90} & \alpha^{130} & \cdots & \alpha^{215} \\ 1 & \alpha^{45} & \alpha^{90} & \alpha^{135} & \alpha^{180} & \alpha^{225} & \alpha^{15} & \alpha^{60} & \alpha^{105} & \alpha^{150} & \alpha^{145} & \alpha^{240} & \alpha^{30} & \alpha^{75} & \alpha^{120} & \alpha^{165} & \alpha^{210} & \cdots & \alpha^{210} \\ 1 & \alpha^{50} & \alpha^{100} & \alpha^{150} & \alpha^{200} & \alpha^{250} & \alpha^{45} & \alpha^{95} & \alpha^{145} & \alpha^{195} & \alpha^{195} & \alpha^{40} & \alpha^{90} & \alpha^{140} & \alpha^{190} & \alpha^{240} & \alpha^{35} & \cdots & \alpha^{205} \\ 1 & \alpha^{55} & \alpha^{110} & \alpha^{165} & \alpha^{220} & \alpha^{20} & \alpha^{75} & \alpha^{130} & \alpha^{185} & \alpha^{240} & \alpha^{245} & \alpha^{95} & \alpha^{150} & \alpha^{205} & \alpha^{5} & \alpha^{60} & \alpha^{115} & \cdots & \alpha^{200} \\ 1 & \alpha^{60} & \alpha^{120} & \alpha^{180} & \alpha^{240} & \alpha^{45} & \alpha^{105} & \alpha^{165} & \alpha^{225} & \alpha^{30} & \alpha^{40} & \alpha^{150} & \alpha^{210} & \alpha^{15} & \alpha^{75} & \alpha^{135} & \alpha^{195} & \cdots & \alpha^{195} \\ 1 & \alpha^{65} & \alpha^{130} & \alpha^{195} & \alpha^{5} & \alpha^{70} & \alpha^{135} & \alpha^{200} & \alpha^{10} & \alpha^{75} & \alpha^{90} & \alpha^{205} & \alpha^{15} & \alpha^{80} & \alpha^{145} & \alpha^{210} & \alpha^{20} & \cdots & \alpha^{190} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{230} & \alpha^{205} & \alpha^{180} & \alpha^{155} & \alpha^{130} & \alpha^{105} & \alpha^{80} & \alpha^{55} & \alpha^{30} & \alpha^{5} & \alpha^{235} & \alpha^{210} & \alpha^{185} & \alpha^{160} & \alpha^{135} & \alpha^{110} & \cdots & \alpha^{25} \end{bmatrix}$$

Selanjutnya, Matriks *parity-check* H memiliki ukuran $(n-k) \times n = 4 \times 51$, dan berfungsi untuk mendeteksi dan mengoreksi kesalahan melalui perhitungan *syndrome*. Matriks H dibentuk menggunakan *sagemath* berdasarkan perpangkatan dari elemen primitif α di $GF(2^8)$. Adapun bentuk matriks H sebagai berikut:

$$H = \begin{bmatrix} 1 & \alpha^{5} & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \cdots & \alpha^{250} \\ 1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \cdots & \alpha^{245} \\ 1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \cdots & \alpha^{240} \\ 1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{160} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \cdots & \alpha^{235} \end{bmatrix}$$

4.2 Proses *Encoding*

Pada proses *encoding* menggunakan algoritma *Reed-Solomon*, misalkan teks pesan ayat Al-Qur'an "وانا لجعلون ما عليها صعيدا جرزا" terdiri dari 31 karakter *Unicode*, yang mencakup huruf hijaiyah beserta spasi sebagai pemisah kata. Setiap karakter terlebih dahulu dikonversi ke dalam representasi *Unicode* 16 −bit, kemudian setiap kode 16 −bit tersebut dipisahkan menjadi dua simbol 8 −bit, sehingga diperoleh total 62 simbol. Dengan menggunakan parameter kode Reed-Solomon adalah $n = 51$, $t = 2$, $k = 47$, blok pesan tersebut dibagi menjadi dua bagian, yaitu blok pesan

pertama M_1 yang memuat 47 simbol dan blok pesan kedua M_2 yang berisi 15 simbol sisanya.

Tabel 4.1 Representasi Ayat Unicode-16 dalam Bentuk Biner

No	Huruf	Biner-16 Bit	No	Huruf	Biner-16 Bit
1	و	0000011001001000	17	ل	0000011001000100
2	ا	0000011000100111	18	ي	0000011001001010
3	ن	0000011001000110	29	ه	0000011001000111
4	ا	0000011000100111	20	ا	0000011000100111
5	ـ	0000000000100000	21	ـ	0000000000100000
6	ل	0000011001000100	22	ص	0000011000110101
7	ج	0000011000101100	23	ع	0000011000111001
8	ع	0000011000111001	24	ي	0000011001001010
9	ل	0000011001000100	25	د	0000011000101111
10	و	0000011001001000	26	ا	0000011000100111
11	ن	0000011001000110	27	ـ	0000000000100000
12	ـ	0000000000100000	28	ج	0000011000101100
13	م	0000011001000101	29	ر	0000011000110001
14	ا	0000011000100111	30	ز	0000011000110010
15	ـ	0000000000100000	31	ا	0000011000100111
16	ع	0000011000111001			

وانا لجعلون ما عليها صعيدا جرزا =

[0000011001001000,0000011000100111,0000011001000110,0000011000100111,0000000000100000,0000011001000100,0000011000101100,0000011000111001,0000011001000100,0000011001001000,0000011001000110,0000000000100000,0000011001000101,0000011000100111,0000000000100000,0000011000111001,0000011001000100,0000011001001010,0000011001000111,0000011000100111,0000000000100000,0000011000110101,0000011000111001,0000011001001010,0000011000101111,0000011000100111,0000000000100000,0000011000101100,0000011000110001,0000011000110010,0000011000100111]

Setiap nilai 16 −bit dipisah menjadi dua bagian 8 −bit:

$$[0000011001001000] = [00000110] \parallel [01001000]$$

Kode biner hasil konversi *Unicode* dibagi menjadi simbol sepanjang 8 bit sesuai dengan panjang elemen pada $GF(2^8)$. Setiap blok pesan memiliki panjang 47 simbol, sesuai dengan $k = 47$.

$$M_1 = \begin{bmatrix} [00000110][01001000][00000110][00100111][00000110][01000110] \\ [00000110][00100111][00000000][00100000][00000110][01000100] \\ [00000110][00101100][00000110][00111001][00000110][01000100] \\ [00000110][01001000][00000110][01000110][00000000][00100000] \\ [00000110][01000101][00000110][00100111][00000000][00100000] \\ [00000110][00111001][00000110][01000100][00000110][01001010] \\ [00000110][01000111][00000110][00100111][00000000][00100000] \\ [00000110][00110101][00000110][00111001][00000110] \end{bmatrix}$$

$$M_2 = \begin{bmatrix} [01001010][00000110][00101111][00000110][00100111][00000000] \\ [00100000][00000110][00101100][00000110][00110001][00000110] \\ [00110010][00000110][00100111][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000] \end{bmatrix}$$

Setiap simbol $8-$bit dipetakan ke elemen $GF(2^8)$ berdasarkan representasi polinomial dari Tabel 2.5.

$$M_1 = [\alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{33}, \alpha^{26}, \alpha^{48}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{154},$$

$$\alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{48}, 0, \alpha^{5}, \alpha^{26}, \alpha^{221}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{154}, \alpha^{26}, \alpha^{102}$$

$$\alpha^{26}, \alpha^{37}, \alpha^{26}, \alpha^{253}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{39}, \alpha^{26}, \alpha^{154}, \alpha^{26}]$$

$$M_2 = [\alpha^{37}, \alpha^{26}, \alpha^{69}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{181}, \alpha^{26}, \alpha^{194}, \alpha^{26}, \alpha^{33}, 0,0,$$

$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

Pada blok kedua (M_2) hanya terdapat 15 simbol representasi polinomial pada $GF(2^8)$. Oleh karena itu, dilakukan penambahan padding 32 simbol bernilai 0 di bagian belakang untuk memenuhi panjang blok menjadi 47 simbol. Dengan demikian, setiap blok pesan memiliki panjang yang seragam, yaitu 47 simbol, sesuai dengan parameter k pada kode Reed-Solomon.

Langkah selanjutnya adalah proses *encoding* menggunakan matriks generator G. Proses *encoding* dilakukan dengan mengalikan vektor pesan $M = [M_0, M_1, \ldots, M_{k-1}]$ dengan matriks generator G berukuran $k \times n$, sehingga menghasilkan vektor kode $C = M \times G$ sepanjang n simbol.

Simbol pertama *codeword* dihitung dari:

$$c_0 = \sum_{i=0}^{46} m_i \cdot G_{i,0}$$

Karena kolom $j = 0$ di mana $G_{i,0} = 1$ sehingga setiap $m_i \cdot G_{i,0} = m_i$ Jadi kolom $0 = [1, 1, 1, 1, \ldots, 1]$ (panjang 47 simbol).

Data pesan M_1 pertama:

$M_1 = [\alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{33}, \alpha^{26}, \alpha^{48}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{154},$

$\alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{48}, 0, \alpha^{5}, \alpha^{26}, \alpha^{221}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{154}, \alpha^{26}, \alpha^{102}$

$\alpha^{26}, \alpha^{37}, \alpha^{26}, \alpha^{253}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{39}, \alpha^{26}, \alpha^{154}, \alpha^{26}]$

Maka:

$$c_0 = \alpha^{26} \cdot 1 \oplus \alpha^{226} \cdot 1 \oplus \alpha^{26} \cdot 1 \oplus \cdots \oplus \alpha^{26} \cdot 1$$

Hasil perhitungan simbol-simbol *codeword* yang diperoleh dari proses *encoding* ditampilkan pada Tabel 4.1.

Tabel 4.2 Tabel Perhitungan XOR untuk Proses *Encoding*

i	m_i	Operasi xor	Hasil	i	m_i	Operasi xor	Hasil
$i = 00$	α^{26}	$0 \oplus \alpha^{26}$	α^{26}	$i = 24$	α^{26}	$\alpha^{248} \oplus \alpha^{26}$	α^{197}
$i = 01$	α^{226}	$\alpha^{26} \oplus \alpha^{226}$	α^{34}	$i = 25$	α^{221}	$\alpha^{197} \oplus \alpha^{221}$	α^{59}
$i = 02$	α^{26}	$\alpha^{34} \oplus \alpha^{26}$	α^{226}	$i = 26$	α^{26}	$\alpha^{51} \oplus \alpha^{26}$	α^{31}
$i = 03$	α^{33}	$\alpha^{226} \oplus \alpha^{33}$	α^{61}	$i = 27$	α^{33}	$\alpha^{31} \oplus \alpha^{33}$	α^{203}
$i = 04$	α^{26}	$\alpha^{61} \oplus \alpha^{26}$	α^{58}	$i = 28$	0	$\alpha^{203} \oplus 0$	α^{203}
$i = 05$	α^{48}	$\alpha^{58} \oplus \alpha^{48}$	α^{69}	$i = 29$	α^{5}	$\alpha^{203} \oplus \alpha^{5}$	α^{178}
$i = 06$	α^{26}	$\alpha^{69} \oplus \alpha^{26}$	α^{147}	$i = 30$	α^{26}	$\alpha^{178} \oplus \alpha^{26}$	α^{37}
$i = 07$	α^{33}	$\alpha^{147} \oplus \alpha^{33}$	α^{199}	$i = 31$	α^{154}	$\alpha^{37} \oplus \alpha^{154}$	α^{123}
$i = 08$	0	$\alpha^{199} \oplus 0$	α^{199}	$i = 32$	α^{26}	$\alpha^{123} \oplus \alpha^{26}$	α^{120}
$i = 09$	α^{5}	$\alpha^{199} \oplus \alpha^{5}$	α^{130}	$i = 33$	α^{102}	$\alpha^{120} \oplus \alpha^{102}$	α^{113}
$i = 10$	α^{26}	$\alpha^{130} \oplus \alpha^{26}$	α^{53}	$i = 34$	α^{26}	$\alpha^{113} \oplus \alpha^{26}$	α^{236}
$i = 11$	α^{102}	$\alpha^{53} \oplus \alpha^{102}$	α^{250}	$i = 35$	α^{37}	$\alpha^{256} \oplus \alpha^{37}$	α^{211}
$i = 12$	α^{26}	$\alpha^{250} \oplus \alpha^{26}$	α^{40}	$i = 36$	α^{26}	$\alpha^{211} \oplus \alpha^{26}$	α^{30}
$i = 13$	α^{240}	$\alpha^{40} \oplus \alpha^{240}$	α^{48}	$i = 37$	α^{253}	$\alpha^{30} \oplus \alpha^{253}$	α^{33}
$i = 14$	α^{26}	$\alpha^{48} \oplus \alpha^{26}$	α^{54}	$i = 38$	α^{26}	$\alpha^{33} \oplus \alpha^{26}$	α^{138}
$i = 15$	α^{154}	$\alpha^{54} \oplus \alpha^{154}$	α^{152}	$i = 39$	α^{33}	$\alpha^{138} \oplus \alpha^{33}$	α^{26}
$i = 16$	α^{26}	$\alpha^{152} \oplus \alpha^{26}$	α^{134}	$i = 40$	0	$\alpha^{26} \oplus 0$	α^{26}
$i = 17$	α^{102}	$\alpha^{134} \oplus \alpha^{102}$	α^{236}	$i = 41$	α^{5}	$\alpha^{26} \oplus \alpha^{5}$	α^{15}
$i = 18$	α^{26}	$\alpha^{236} \oplus \alpha^{26}$	α^{218}	$i = 42$	α^{26}	$\alpha^{15} \oplus \alpha^{26}$	α^{5}
$i = 19$	α^{226}	$\alpha^{218} \oplus \alpha^{226}$	α^{184}	$i = 43$	α^{39}	$\alpha^{5} \oplus \alpha^{39}$	α^{141}
$i = 20$	α^{26}	$\alpha^{184} \oplus \alpha^{26}$	α^{140}	$i = 44$	α^{26}	$\alpha^{141} \oplus \alpha^{26}$	α^{14}
$i = 21$	α^{48}	$\alpha^{140} \oplus \alpha^{48}$	α^{181}	$i = 45$	α^{154}	$\alpha^{14} \oplus \alpha^{154}$	α^{142}
$i = 22$	0	$\alpha^{181} \oplus 0$	α^{181}	$i = 46$	α^{26}	$\alpha^{142} \oplus \alpha^{26}$	α^{240}
$i = 23$	α^{5}	$\alpha^{181} \oplus \alpha^{5}$	α^{248}				

Sehingga didapatkan simbol vektor *codeword* pertama adalah α^{240}.

$C_1 = M_1 \times G =$

$= [\alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{33}, \alpha^{26}, \alpha^{48}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{154},$

$\alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{48}, 0, \alpha^{5}, \alpha^{26}, \alpha^{221}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{154}, \alpha^{26}, \alpha^{102}$

$\alpha^{26}, \alpha^{37}, \alpha^{26}, \alpha^{253}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{39}, \alpha^{26}, \alpha^{154}, \alpha^{26}] \times$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \alpha^{5} & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \alpha^{65} & \alpha^{70} & \alpha^{75} & \alpha^{80} & \cdots & \alpha^{250} \\
1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \alpha^{130} & \alpha^{140} & \alpha^{150} & \alpha^{160} & \cdots & \alpha^{245} \\
1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \alpha^{195} & \alpha^{210} & \alpha^{222} & \alpha^{240} & \cdots & \alpha^{240} \\
1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{140} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \alpha^{5} & \alpha^{25} & \alpha^{45} & \alpha^{65} & \cdots & \alpha^{235} \\
1 & \alpha^{25} & \alpha^{50} & \alpha^{75} & \alpha^{100} & \alpha^{125} & \alpha^{150} & \alpha^{175} & \alpha^{200} & \alpha^{225} & \alpha^{250} & \alpha^{20} & \alpha^{45} & \alpha^{70} & \alpha^{95} & \alpha^{120} & \alpha^{145} & \cdots & \alpha^{230} \\
1 & \alpha^{30} & \alpha^{60} & \alpha^{90} & \alpha^{120} & \alpha^{150} & \alpha^{180} & \alpha^{210} & \alpha^{240} & \alpha^{15} & \alpha^{45} & \alpha^{75} & \alpha^{105} & \alpha^{135} & \alpha^{165} & \alpha^{195} & \alpha^{225} & \cdots & \alpha^{225} \\
1 & \alpha^{35} & \alpha^{70} & \alpha^{105} & \alpha^{140} & \alpha^{175} & \alpha^{210} & \alpha^{245} & \alpha^{25} & \alpha^{60} & \alpha^{95} & \alpha^{130} & \alpha^{165} & \alpha^{200} & \alpha^{235} & \alpha^{15} & \alpha^{50} & \cdots & \alpha^{220} \\
1 & \alpha^{40} & \alpha^{80} & \alpha^{120} & \alpha^{160} & \alpha^{200} & \alpha^{240} & \alpha^{25} & \alpha^{65} & \alpha^{105} & \alpha^{9} & \alpha^{185} & \alpha^{225} & \alpha^{10} & \alpha^{50} & \alpha^{90} & \alpha^{130} & \cdots & \alpha^{215} \\
1 & \alpha^{45} & \alpha^{90} & \alpha^{135} & \alpha^{180} & \alpha^{225} & \alpha^{15} & \alpha^{60} & \alpha^{105} & \alpha^{150} & \alpha^{145} & \alpha^{240} & \alpha^{30} & \alpha^{75} & \alpha^{120} & \alpha^{165} & \alpha^{210} & \cdots & \alpha^{210} \\
1 & \alpha^{50} & \alpha^{100} & \alpha^{150} & \alpha^{200} & \alpha^{250} & \alpha^{45} & \alpha^{95} & \alpha^{145} & \alpha^{195} & \alpha^{195} & \alpha^{40} & \alpha^{90} & \alpha^{140} & \alpha^{190} & \alpha^{240} & \alpha^{35} & \cdots & \alpha^{205} \\
1 & \alpha^{55} & \alpha^{110} & \alpha^{165} & \alpha^{220} & \alpha^{20} & \alpha^{75} & \alpha^{130} & \alpha^{185} & \alpha^{240} & \alpha^{245} & \alpha^{95} & \alpha^{150} & \alpha^{205} & \alpha^{5} & \alpha^{60} & \alpha^{115} & \cdots & \alpha^{200} \\
1 & \alpha^{60} & \alpha^{120} & \alpha^{180} & \alpha^{240} & \alpha^{45} & \alpha^{105} & \alpha^{165} & \alpha^{225} & \alpha^{30} & \alpha^{40} & \alpha^{150} & \alpha^{210} & \alpha^{15} & \alpha^{75} & \alpha^{135} & \alpha^{195} & \cdots & \alpha^{195} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha^{230} & \alpha^{205} & \alpha^{180} & \alpha^{155} & \alpha^{130} & \alpha^{105} & \alpha^{80} & \alpha^{55} & \alpha^{30} & \alpha^{5} & \alpha^{235} & \alpha^{210} & \alpha^{185} & \alpha^{160} & \alpha^{135} & \alpha^{110} & \cdots & \alpha^{25}
\end{bmatrix}$$

$C_1 = [\alpha^{240}, \alpha^{35}, \alpha^{39}, \alpha^{120}, \alpha^{153}, \alpha^{54}, \alpha^{227}, \alpha^{151}, \alpha^{233}, \alpha^{198}, \alpha^{85}, \alpha^{213}, \alpha^{44}, \alpha^{104}, \alpha^{3},$

$\alpha^{50}, \alpha^{26}, \alpha^{251}, \alpha^{173}, \alpha^{65}, \alpha^{39}, \alpha^{32}, \alpha^{25}, \alpha^{231}, \alpha^{133}, \alpha^{110}, \alpha^{113}, \alpha^{1}, \alpha^{43}, \alpha^{210}, \alpha^{151},$

$\alpha^{144}, \alpha^{11}, \alpha^{0}, \alpha^{174}, \alpha^{135}, \alpha^{84}, \alpha^{15}, \alpha^{124}, \alpha^{96}, \alpha^{18}, \alpha^{160}, \alpha^{127}, \alpha^{228}, \alpha^{178}, \alpha^{22}, \alpha^{216}, \alpha^{181},$

$\alpha^{52}, \alpha^{40}, \alpha^{196}]$

$C_2 = M_2 \times G =$

$= [\alpha^{37}, \alpha^{26}, \alpha^{69}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{181}, \alpha^{26}, \alpha^{194}, \alpha^{26}, \alpha^{33}, 0,0$

$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] \times$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \alpha^{5} & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \alpha^{65} & \alpha^{70} & \alpha^{75} & \alpha^{80} & \cdots & \alpha^{250} \\
1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \alpha^{130} & \alpha^{140} & \alpha^{150} & \alpha^{160} & \cdots & \alpha^{245} \\
1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \alpha^{195} & \alpha^{210} & \alpha^{222} & \alpha^{240} & \cdots & \alpha^{240} \\
1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{140} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \alpha^{5} & \alpha^{25} & \alpha^{45} & \alpha^{65} & \cdots & \alpha^{235} \\
1 & \alpha^{25} & \alpha^{50} & \alpha^{75} & \alpha^{100} & \alpha^{125} & \alpha^{150} & \alpha^{175} & \alpha^{200} & \alpha^{225} & \alpha^{250} & \alpha^{20} & \alpha^{45} & \alpha^{70} & \alpha^{95} & \alpha^{120} & \alpha^{145} & \cdots & \alpha^{230} \\
1 & \alpha^{30} & \alpha^{60} & \alpha^{90} & \alpha^{120} & \alpha^{150} & \alpha^{180} & \alpha^{210} & \alpha^{240} & \alpha^{15} & \alpha^{45} & \alpha^{75} & \alpha^{105} & \alpha^{135} & \alpha^{165} & \alpha^{195} & \alpha^{225} & \cdots & \alpha^{225} \\
1 & \alpha^{35} & \alpha^{70} & \alpha^{105} & \alpha^{140} & \alpha^{175} & \alpha^{210} & \alpha^{245} & \alpha^{25} & \alpha^{60} & \alpha^{95} & \alpha^{130} & \alpha^{165} & \alpha^{200} & \alpha^{235} & \alpha^{15} & \alpha^{50} & \cdots & \alpha^{220} \\
1 & \alpha^{40} & \alpha^{80} & \alpha^{120} & \alpha^{160} & \alpha^{200} & \alpha^{240} & \alpha^{25} & \alpha^{65} & \alpha^{105} & \alpha^{9} & \alpha^{185} & \alpha^{225} & \alpha^{10} & \alpha^{50} & \alpha^{90} & \alpha^{130} & \cdots & \alpha^{215} \\
1 & \alpha^{45} & \alpha^{90} & \alpha^{135} & \alpha^{180} & \alpha^{225} & \alpha^{15} & \alpha^{60} & \alpha^{105} & \alpha^{150} & \alpha^{145} & \alpha^{240} & \alpha^{30} & \alpha^{75} & \alpha^{120} & \alpha^{165} & \alpha^{210} & \cdots & \alpha^{210} \\
1 & \alpha^{50} & \alpha^{100} & \alpha^{150} & \alpha^{200} & \alpha^{250} & \alpha^{45} & \alpha^{95} & \alpha^{145} & \alpha^{195} & \alpha^{195} & \alpha^{40} & \alpha^{90} & \alpha^{140} & \alpha^{190} & \alpha^{240} & \alpha^{35} & \cdots & \alpha^{205} \\
1 & \alpha^{55} & \alpha^{110} & \alpha^{165} & \alpha^{220} & \alpha^{20} & \alpha^{75} & \alpha^{130} & \alpha^{185} & \alpha^{240} & \alpha^{245} & \alpha^{95} & \alpha^{150} & \alpha^{205} & \alpha^{5} & \alpha^{60} & \alpha^{115} & \cdots & \alpha^{200} \\
1 & \alpha^{60} & \alpha^{120} & \alpha^{180} & \alpha^{240} & \alpha^{45} & \alpha^{105} & \alpha^{165} & \alpha^{225} & \alpha^{30} & \alpha^{40} & \alpha^{150} & \alpha^{210} & \alpha^{15} & \alpha^{75} & \alpha^{135} & \alpha^{195} & \cdots & \alpha^{195} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha^{230} & \alpha^{205} & \alpha^{180} & \alpha^{155} & \alpha^{130} & \alpha^{105} & \alpha^{80} & \alpha^{55} & \alpha^{30} & \alpha^{5} & \alpha^{235} & \alpha^{210} & \alpha^{185} & \alpha^{160} & \alpha^{135} & \alpha^{110} & \cdots & \alpha^{25}
\end{bmatrix}$$

$C_2 = [\alpha^{40}, \alpha^{243}, \alpha^{86}, \alpha^{119}, \alpha^{46}, \alpha^{75}, \alpha^{106}, \alpha^{116}, \alpha^{203}, \alpha^{91}, \alpha^{133}, \alpha^{21}, \alpha^{175}, \alpha^{8}, \alpha^{112}, \alpha^{21},$
$\alpha^{27}, \alpha^{231}, \alpha^{241}, \alpha^{22}, \alpha^{205}, \alpha^{216}, \alpha^{22}, \alpha^{84}, \alpha^{101}, \alpha^{246}, \alpha^{67}, \alpha^{30}, \alpha^{184}, \alpha^{96}, \alpha^{28}, \alpha^{226}, \alpha^{219}, \alpha^{25},$
$\alpha^{118}, \alpha^{141}, \alpha^{251}, \alpha^{182}, \alpha^{49}, \alpha^{181}, \alpha^{203}, \alpha^{46}, \alpha^{49}, \alpha^{162}, \alpha^{41}, \alpha^{113}, \alpha^{63}, \alpha^{199}, \alpha^{239}, \alpha^{72}, \alpha^{226}]$

4.3 Transmisi (Penambahan Polinomial *Error*)

Diasumsikan pada *codeword* pertama terjadi dua error, posisi *error* masing-masing berlokasi pada posisi [5,20]. Nilai v_1 kemudian diperoleh melalui proses perhitungan secara komputasi yaitu,

$$v_1 = C_1 + e$$

$v_1 = [\alpha^{240}, \alpha^{35}, \alpha^{39}, \alpha^{120}, \alpha^{153}, \alpha^{54}, \alpha^{227}, \alpha^{151}, \alpha^{233}, \alpha^{198}, \alpha^{85}, \alpha^{213}, \alpha^{44}, \alpha^{104}, \alpha^{3}, \alpha^{50},$
$\alpha^{26}, \alpha^{251}, \alpha^{173}, \alpha^{65}, \alpha^{39}, \alpha^{32}, \alpha^{25}, \alpha^{231}, \alpha^{133}, \alpha^{110}, \alpha^{113}, \alpha^{1}, \alpha^{43}, \alpha^{210}, \alpha^{151}, \alpha^{144}, \alpha^{11}, \alpha^{0},$
$\alpha^{174}, \alpha^{135}, \alpha^{84}, \alpha^{15}, \alpha^{124}, \alpha^{96}, \alpha^{18}, \alpha^{160}, \alpha^{127}, \alpha^{228}, \alpha^{178}, \alpha^{22}, \alpha^{216}, \alpha^{181}, \alpha^{52}, \alpha^{40}, \alpha^{196}]$

$+ [0x^{50} + 0x^{49} + 0x^{48} + 0x^{47} + 0x^{46} + \boldsymbol{\alpha^{146}x^{45}} + 0x^{44} + 0x^{43} + 0x^{42} +$
$0x^{41}0x^{40} + 0x^{39} + 0x^{38} + 0x^{37} + 0x^{36} + 0x^{35} + 0x^{34} + 0x^{33} + 0x^{32} + 0x^{31} +$
$\boldsymbol{\alpha^{233}x^{30}} + 0x^{29} + 0x^{28} + 0x^{27}+0x^{26}+0x^{25}+0x^{24} + 0x^{23} + 0x^{22} + 0x^{21}+0x^{20} +$
$0x^{19} + 0x^{18} + 0x^{17} + 0x^{16} + 0x^{15} + 0x^{14} + 0x^{13} + 0x^{12} + 0x^{11} + 0x^{10} + 0x^{9} +$
$0x^{8} + 0x^{7} + 0x^{6} + 0x^{5} + 0x^{4} + 0x^{3} + 0x^{2} + 0x^{1} + 0] =$

$v_1 =$

$[\alpha^{240}, \alpha^{35}, \alpha^{39}, \alpha^{120}, \alpha^{153}, \boldsymbol{\alpha^{73}}, \alpha^{227}, \alpha^{151}, \alpha^{233}, \alpha^{198}, \alpha^{85}, \alpha^{213}, \alpha^{44}, \alpha^{104}, \alpha^{3}, \alpha^{50}, \alpha^{26},$
$\alpha^{251}, \alpha^{173}, \alpha^{65}, \boldsymbol{\alpha^{164}}, \alpha^{32}, \alpha^{25}, \alpha^{231}, \alpha^{133}, \alpha^{110}, \alpha^{113}, \alpha^{1}, \alpha^{43}, \alpha^{210}, \alpha^{151}, \alpha^{144}, \alpha^{11}, \alpha^{0},$
$\alpha^{174}, \alpha^{135}, \alpha^{84}, \alpha^{15}, \alpha^{124}, \alpha^{96}, \alpha^{18}, \alpha^{160}, \alpha^{127}, \alpha^{228}, \alpha^{178}, \alpha^{22}, \alpha^{216}, \alpha^{181}, \alpha^{52}, \alpha^{40}, \alpha^{196}]$

Pada *codeword* kedua, posisi *error* masing-masing berlokasi pada posisi [10,35] yaitu,

$$v_2 = C_2 + e$$

$v_2 = [\alpha^{40}, \alpha^{243}, \alpha^{86}, \alpha^{119}, \alpha^{46}, \alpha^{75}, \alpha^{106}, \alpha^{116}, \alpha^{203}, \alpha^{91}, \alpha^{133}, \alpha^{21}, \alpha^{175}, \alpha^{8}, \alpha^{112}, \alpha^{21}, \alpha^{27},$
$\alpha^{231}, \alpha^{241}, \alpha^{22}, \alpha^{205}, \alpha^{216}, \alpha^{22}, \alpha^{84}, \alpha^{101}, \alpha^{246}, \alpha^{67}, \alpha^{30}, \alpha^{184}, \alpha^{96}, \alpha^{28}, \alpha^{226}, \alpha^{219}, \alpha^{25},$
$\alpha^{118}, \alpha^{141}, \alpha^{251}, \alpha^{182}, \alpha^{49}, \alpha^{181}, \alpha^{203}, \alpha^{46}, \alpha^{49}, \alpha^{162}, \alpha^{41}, \alpha^{113}, \alpha^{63}, \alpha^{199}, \alpha^{239}, \alpha^{72}, \alpha^{226}] +$
$+ [0x^{50} + 0x^{49} + 0x^{48} + 0x^{47} + 0x^{46} + 0x^{45} + 0x^{44} + 0x^{43} + 0x^{42} + 0x^{41} +$
$\alpha^{188}x^{40} + 0x^{39} + 0x^{38} + 0x^{37} + 0x^{36} + 0x^{35} + 0x^{34} + 0x^{33} + 0x^{32} + 0x^{31} +$
$0x^{30} + 0x^{29} + 0x^{28} + 0x^{27} + 0x^{26} + 0x^{25} + 0x^{24} + 0x^{23} + 0x^{22} + 0x^{21} + 0x^{20} + 0x^{19} +$
$0x^{18} + 0x^{17} + 0x^{16} + \alpha^{186}x^{15} + 0x^{14} + 0x^{13} + 0x^{12} + 0x^{11} + 0x^{10} + 0x^{9} + 0x^{8} +$
$0x^{7} + 0x^{6} + 0x^{5} + 0x^{4} + 0x^{3} + 0x^{2} + 0x^{1} + 0] =$
$v_2 = [\alpha^{40}, \alpha^{243}, \alpha^{86}, \alpha^{119}, \alpha^{46}, \alpha^{75}, \alpha^{106}, \alpha^{116}, \alpha^{203}, \alpha^{91}, \boldsymbol{\alpha^{196}}, \alpha^{21}, \alpha^{175}, \alpha^{8}, \alpha^{112}, \alpha^{21}, \alpha^{27},$
$\alpha^{231}, \alpha^{241}, \alpha^{22}, \alpha^{205}, \alpha^{216}, \alpha^{22}, \alpha^{84}, \alpha^{101}, \alpha^{246}, \alpha^{67}, \alpha^{30}, \alpha^{184}, \alpha^{96}, \alpha^{28}, \alpha^{226}, \alpha^{219}, \alpha^{25},$
$\alpha^{118}, \boldsymbol{\alpha^{172}}, \alpha^{251}, \alpha^{182}, \alpha^{49}, \alpha^{181}, \alpha^{203}, \alpha^{46}, \alpha^{49}, \alpha^{162}, \alpha^{41}, \alpha^{113}, \alpha^{63}, \alpha^{199}, \alpha^{239}, \alpha^{72}, \alpha^{226}]$

4.4 Proses *Decoding*

4.4.1 Proses Deteksi

Sindrom adalah bagian utama dalam proses *decoding* Reed-Solomon karena digunakan untuk mendeteksi adanya kesalahan (*error*).

$$v = v_0 + v_1 + v_2 + \cdots + v_{n-1},$$

dan α elemen primitif $GF(2^8)$, maka sindrom ke-j:

$$S_j = v(\alpha^j) \sum_{i=0}^{n-1} v_i \cdot (\alpha^{5(0+i)})^j, j = 1, 2, \ldots, 2t.$$

$$S_j = \sum_{i=0}^{50} v_i \cdot (\alpha^{5i})^j, j = 1, 2,3,4. \text{ Atau,}$$

$$S = H \times v_i^T$$

1. Hitung $S_1 = v_1$

   Dalam perhitungan sindrom bisa juga menggunakan rumus

$$S_1 = H \times v_1^T$$

$$S_1 = \begin{bmatrix} 1 & \alpha^5 & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \cdots & \alpha^{250} \\ 1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \cdots & \alpha^{245} \\ 1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \cdots & \alpha^{240} \\ 1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{160} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \cdots & \alpha^{235} \end{bmatrix} \times \begin{bmatrix} \alpha^{240} \\ \alpha^{35} \\ \alpha^{39} \\ \alpha^{120} \\ \alpha^{153} \\ \alpha^{73} \\ \alpha^{227} \\ \alpha^{151} \\ \alpha^{233} \\ \alpha^{198} \\ \alpha^{85} \\ \alpha^{213} \\ \alpha^{44} \\ \alpha^{104} \\ \alpha^{3} \\ \alpha^{50} \\ \alpha^{26} \\ \alpha^{251} \\ \alpha^{173} \\ \alpha^{65} \\ \alpha^{39} \\ \alpha^{32} \\ \alpha^{25} \\ \vdots \\ \alpha^{196} \end{bmatrix}$$

$$S_1 = \begin{bmatrix} \alpha^{236} \\ \alpha^{163} \\ \alpha^{49} \\ \alpha^{51} \end{bmatrix}$$

$$S_1(x) = s_0 + s_1 x + s_2 x^2 + s_3 x^3.$$

$$S_1(x) = \alpha^{236} + \alpha^{163} x + \alpha^{49} x^2 + \alpha^{51} x^3.$$

2. Hitung $S_2 = v_2$

$$S_2 = H \times v_2^T$$

$$S_2 = \begin{bmatrix} 1 & \alpha^5 & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} & \alpha^{35} & \alpha^{40} & \alpha^{45} & \alpha^{50} & \alpha^{55} & \alpha^{60} & \cdots & \alpha^{250} \\ 1 & \alpha^{10} & \alpha^{20} & \alpha^{30} & \alpha^{40} & \alpha^{50} & \alpha^{60} & \alpha^{70} & \alpha^{80} & \alpha^{90} & \alpha^{100} & \alpha^{110} & \alpha^{120} & \cdots & \alpha^{245} \\ 1 & \alpha^{15} & \alpha^{30} & \alpha^{45} & \alpha^{60} & \alpha^{75} & \alpha^{90} & \alpha^{105} & \alpha^{120} & \alpha^{135} & \alpha^{150} & \alpha^{165} & \alpha^{180} & \cdots & \alpha^{240} \\ 1 & \alpha^{20} & \alpha^{40} & \alpha^{60} & \alpha^{80} & \alpha^{100} & \alpha^{120} & \alpha^{140} & \alpha^{160} & \alpha^{180} & \alpha^{200} & \alpha^{220} & \alpha^{240} & \cdots & \alpha^{235} \end{bmatrix} \times \begin{bmatrix} \alpha^{40} \\ \alpha^{243} \\ \alpha^{86} \\ \alpha^{119} \\ \alpha^{46} \\ \alpha^{75} \\ \alpha^{106} \\ \alpha^{116} \\ \alpha^{203} \\ \alpha^{91} \\ \alpha^{196} \\ \alpha^{21} \\ \alpha^{175} \\ \alpha^{8} \\ \alpha^{112} \\ \alpha^{21} \\ \alpha^{27} \\ \alpha^{231} \\ \alpha^{241} \\ \alpha^{22} \\ \alpha^{205} \\ \alpha^{216} \\ \alpha^{22} \\ \alpha^{84} \\ \vdots \\ \alpha^{226} \end{bmatrix}$$

$$S_2 = \begin{bmatrix} \alpha^{166} \\ \alpha^{138} \\ \alpha^{247} \\ \alpha^{248} \end{bmatrix}$$

$$S_1(x) = s_0 + s_1 x + s_2 x^2 + s_3 x^3.$$

$$S_1(x) = \alpha^{166} + \alpha^{138} x + \alpha^{247} x^2 + \alpha^{248} x^3.$$

4.4.2 Menentukan Polonomial Lokasi Kesalahan

Menentukan polynomial locator *error* menggunakan metode algoritma *Berlekamp–Massey*.

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$$

1. Diketahui v_1

   $$S_1 = [s_0, s_1, s_2, s_3] = [\alpha^{236}, \alpha^{163}, \alpha^{49}, \alpha^{51}]$$

   Inisialisasi

   $C(x) = [c_0 + c_1 x + c_2 x^2]$, polinomial penentu kesalahan $\sigma(x)$

   $B(x) =$ Polinomial koneksi cadangan

   $L =$ Derajat polinomial penentu kesalahan $C(x)$

   $m =$ Selisih iterasi sejak terakhir kali L diperbarui

   $b \ =$ Pembagi normalisasi

   $d =$ *Discrepancy* (ketidaksesuaian) pada iterasi ke-n

   $S_i = [s_0, s_1, s_2, s_3]$ Sindrom ke-i

   a. Iterasi $ke - 0$ $(n = 0)$

      $C(x) = 1$

      $B(x) = 1$

      $L \quad = 0$

      $m \quad = 1$

$b \quad = 1$

$$d = \sum_{i=0}^{L} c_i s_{n-i}$$

$$d = c_0 s_0 = 1 \cdot s_0 = \alpha^{236}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x)$

$C(x) = 1 \oplus \frac{\alpha^{236}}{1} x \cdot 1 = 1 + \alpha^{236} x$

Karena $2L \leq n$ $(0 \leq 0)$, maka:

$C(x) = 1 + \alpha^{236} x$

$L \quad = 1$

$B(x) = 1$

$b \quad = \alpha^{236}$

$m \quad = 1$

b. Iterasi $ke - 1$ $(n = 1)$

$C(x) = 1 + \alpha^{236} x$

$L \quad = 1$

$B(x) = 1$

$b \quad = \alpha^{236}$

$m \quad = 1$

$$d = c_0 s_1 + c_1 s_0$$

$$d = s_1 + c_1 \cdot s_0$$

$$d = \alpha^{163} \oplus (\alpha^{236} \cdot \alpha^{236})$$

$$d = \alpha^{163} \oplus \alpha^{217} = \alpha^{116}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x)$

$= (1 + \alpha^{236}) \oplus \frac{\alpha^{116}}{\alpha^{236}} x \cdot 1$

$= (1 + \alpha^{236}) \oplus \alpha^{135} x$

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135}) x$

Karena $2L \leq n$ ($2 \leq 1$ *false*) tidak terpenuhi, maka:

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135}) x$

$L \quad = 1$, tetap

$B(x) = 1$

$b \quad = \alpha^{236}$

$m + 1 = 2$

c. Iterasi $ke - 2$ $(n = 2)$

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135}) x$

$L \quad = 1$

$B(x) = 1$

$b \quad = \alpha^{236}$

$m \quad = 2$

$$d = c_0 s_2 + c_1 s_1 + c_2 s_0$$

$$d = s_2 + c_1 \cdot s_1 + 0 \cdot s_0$$

$$d = \alpha^{49} \oplus ((\alpha^{236} \oplus \alpha^{135}) \cdot \alpha^{163}) \oplus (0 \cdot \alpha^{236})$$

$$d = \alpha^{49} \oplus \left((\alpha^{236} \oplus \alpha^{135}) \cdot \alpha^{163}\right)$$

$$d = \alpha^{49} \oplus \left((\alpha^{236} \cdot \alpha^{163})(\alpha^{135} \oplus \alpha^{163})\right)$$

$$d = \alpha^{49} \oplus \alpha^{144} \oplus \alpha^{43} = \alpha^{206}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x)$

$= (1 + (\alpha^{236} \oplus \alpha^{135})x) \oplus \frac{\alpha^{206}}{\alpha^{236}} x^2 \cdot 1$

$= (1 + (\alpha^{236} \oplus \alpha^{135})x) \oplus \alpha^{225} x^2$

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x + \alpha^{225} x^2$

Karena $2L \leq n$ $(2 \leq 2)$ terpenuhi, maka:

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x + \alpha^{225} x^2$

$L \quad = n + 1 - L = 2$

$B(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x$

$b \quad = \alpha^{206}$

$m \quad = 1$

d. Iterasi $ke - 3$ $(n = 3)$

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x + \alpha^{225} x^2$

$L \quad = 2$

$B(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x$

$b \quad = \alpha^{206}$

$m \quad = 1$

$$d = c_0 s_3 + c_1 s_2 + c_2$$

$$d = s_3 + c_1 \cdot s_2 + c_2 \cdot s_1$$

$$d = \alpha^{51} \oplus \left((\alpha^{236} \oplus \alpha^{135}) \cdot \alpha^{49}\right) \oplus (\alpha^{225} \cdot \alpha^{163})$$

$$d = \alpha^{51} \oplus \left((\alpha^{236} \oplus \alpha^{135}) \cdot \alpha^{49}\right)$$

$$d = \alpha^{51} \oplus (\alpha^{236} \cdot \alpha^{49}) \oplus (\alpha^{135} \cdot \alpha^{49}) \oplus \alpha^{133}$$

$$d = \alpha^{51} \oplus (\alpha^{30} \oplus \alpha^{184}) \oplus \alpha^{133} = \alpha^{153}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) - \frac{153}{206} x^m \cdot B(x)$

$C(x) = (1 + (\alpha^{236} \oplus \alpha^{135})x + \alpha^{225}x^2) \oplus \alpha^{202}x \cdot (1 +$

$(\alpha^{236} \oplus \alpha^{135})x)$

$C(x) = (1 + (\alpha^{236} \oplus \alpha^{135})x + \alpha^{225}x^2) \oplus (\alpha^{202}x + \alpha^{202}(\alpha^{236} \oplus$

$\alpha^{135})x^2)$

$$\mathrm{C_{1new}} = \mathrm{c_1} \oplus \alpha^{202} = (\alpha^{236} \oplus \alpha^{135}) \oplus \alpha^{202} = \alpha^{224}$$

$$\mathrm{C_{2new}} = \mathrm{c_2} \oplus (\alpha^{202}\mathrm{B_1}) = \alpha^{225} \oplus (\alpha^{202}(\alpha^{236} \oplus \alpha^{135}))$$

$$\mathrm{C_{2new}} = \alpha^{225} \oplus (\alpha^{183} \oplus \alpha^{82}) = \alpha^{125}$$

Jadi hasil akhir:

$C(x) = 1 + \alpha^{224}x + \alpha^{125}x^2$

Maka

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2, \qquad \sigma_1 = \alpha^{224}, \sigma_2 = \alpha^{125}.$$

2. Diketahui v_2

$$\mathrm{S_2} = [s_0, s_1, s_2, s_3] = [\alpha^{166}, \alpha^{138}, \alpha^{247}, \alpha^{248}]$$

a. Iterasi $ke - 0$ $(n = 0)$

$C(x) = 1$

$B(x) = 1$

$L = 0$

$m = 1$

$b = 1$

$$d = \sum_{i=0}^{L} c_i s_{n-i}$$

$$d = c_0 s_0 = 1 \cdot s_0 = \alpha^{166}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x) = 1 \oplus \frac{\alpha^{166}}{1} x \cdot 1 = 1 + \alpha^{166} x$

Karena $2L \leq n$ $(0 \leq 0)$, maka:

$C(x) = 1 + \alpha^{166} x$

$L = 1$

$B(x) = 1$

$b = \alpha^{166}$

$m = 1$

b. Iterasi $ke - 1$ $(n = 1)$

$C(x) = 1 + \alpha^{166} x$

$L = 1$

$B(x) = T(x) = 1$

$b = \alpha^{166}$

$m = 1$

$$d = c_0 s_1 + c_1 s_0$$

$$d = s_1 + c_1 \cdot s_0 = \alpha^{138} \oplus (\alpha^{166} \cdot \alpha^{166}) = \alpha^{138} \oplus \alpha^{77} = \alpha^{8}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x) = (1 + \alpha^{166}x) \oplus \frac{\alpha^8}{\alpha^{166}} x \cdot 1 =$

$(1 + \alpha^{166}) \oplus \alpha^{97}x$

$C(x) = 1 + (\alpha^{236} \oplus \alpha^{135})x = 1 + \alpha^{227}x$

Karena $2L \leq n$ tidak terpenuhi ($2 \leq 1$ *false*), maka:

$C(x) = 1 + \alpha^{227}x$

$L = 1$, tetap

$B(x) = 1$

$b = \alpha^{166}$

$m + 1 = 2$

c. Iterasi $ke - 2$ $(n = 2)$

$C(x) = 1 + \alpha^{227}x$

$L = 1$

$B(x) = 1$

$b = \alpha^{166}$

$m = 2$

$$d = c_0 s_2 + c_1 s_1 + c_2 s_0$$

$$d = s_2 + c_1 \cdot s_1 + 0 \cdot s_0$$

$$d = \alpha^{247} \oplus (\alpha^{227} \cdot \alpha^{163}) \oplus 0$$

$$d = \alpha^{247} \oplus \alpha^{(227+138) \bmod 255} = \alpha^{247} \oplus \alpha^{110}$$

$$d = \alpha^{156}$$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x)$

$= (1 + \alpha^{227} x) \oplus \frac{\alpha^{156}}{\alpha^{166}} x^2 \cdot 1$

$= (1 + \alpha^{227} x) \oplus \alpha^{245} x^2$

$C(x) = 1 + \alpha^{227} x + \alpha^{245} x^2$

Karena $2L \leq n$ terpenuhi $(2 \leq 2)$, maka:

$C(x) = 1 + \alpha^{227} x + \alpha^{245} x^2$

$L \quad = n + 1 - L = 2$

$B(x) = 1 + \alpha^{227} x$

$b \quad = \alpha^{156}$

$m \quad = 1$

d. Iterasi $ke - 3$ $(n = 3)$

$C(x) = 1 + \alpha^{227} x + \alpha^{245} x^2$

$L \quad = n + 1 - L = 2$

$B(x) = 1 + \alpha^{227} x$

$b \quad = \alpha^{156}$

$m \quad = 1$

$d = c_0 s_3 + c_1 s_2 + c_2 s_1$

$= s_3 + c_1 \cdot s_2 + c_2 \cdot s_1$

$= \alpha^{248} \oplus (\alpha^{227} \cdot \alpha^{247}) \oplus (\alpha^{245} \cdot \alpha^{138})$

$= \alpha^{248} \oplus \alpha^{219} \oplus \alpha^{128} = \alpha^{196}$

Karena $d \neq 0$

$T(x) = C(x)$

$C(x) = C(x) \oplus \frac{d}{b} x^m \cdot B(x)$

$$= (1 + \alpha^{227}x + \alpha^{245}x^2) \oplus \frac{196}{156}x \cdot (1 + \alpha^{227}x)$$

$$C(x) = (1 + \alpha^{227}x + \alpha^{245}x^2) \oplus \alpha^{40}x \cdot (1 + \alpha^{227}x)$$

$$C(x) = (1 + \alpha^{227}x + \alpha^{245}x^2) \oplus (\alpha^{40}x + \alpha^{12}x^2)$$

$$C_{1new} = \alpha^{227} \oplus \alpha^{202} = \alpha^{244}$$

$$C_{2new} = \alpha^{245} \oplus \alpha^{12} = \alpha^{225}$$

Jadi hasil akhir:

$$C(x) = 1 + \alpha^{244}x + \alpha^{225}x^2$$

Maka

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2, \qquad \sigma_1 = \alpha^{244}, \sigma_2 = \alpha^{225}.$$

4.4.3 Menentukan Posisi Kesalahan (*Error Positions*)

1. v_1

   Polynomial locator $\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

   $$\sigma(x) = 1 + \alpha^{224}x + \alpha^{125}x^2$$

   $\sigma_0 = 1$

   $\sigma_1 = \alpha^{224}$

   $\sigma_2 = \alpha^{125}$

   $$x = \sigma(\alpha^{-e}) = 0$$

   $$\sigma(\alpha^{-e}) = 1 + \alpha^{224}\alpha^{-e} + \alpha^{125}\alpha^{-2e}$$

   Untuk setiap $e = 0,1,2, \ldots, 254$

   $\sigma(\alpha^{-e}) = 1 + \alpha^{224}\alpha^{-e} + \alpha^{125}\alpha^{-2e}$

   $\sigma(\alpha^{-25}) = 1 + \alpha^{224}\alpha^{-25} + \alpha^{125}\alpha^{-2\cdot 25}$

   $\sigma(\alpha^{-25}) = 1 + \alpha^{224-25} + \alpha^{125-50}$

   $\sigma(\alpha^{-25}) = 1 + \alpha^{199} + \alpha^{75}$

$\sigma(\alpha^{-25}) = 0$

$\sigma(\alpha^{-e}) \quad = 1 + \alpha^{224}\alpha^{-e} + \alpha^{125}\alpha^{-2e}$

$\sigma(\alpha^{-100}) = 1 + \alpha^{224}\alpha^{-100} + \alpha^{125}\alpha^{-2\cdot 100}$

$\sigma(\alpha^{-100}) = 1 + \alpha^{224-100} + \alpha^{125-200}$

$\sigma(\alpha^{-100}) = 1 + \alpha^{124} + \alpha^{180}$

$\sigma(\alpha^{-100}) = 0$

Jadi lokasi kesalahan pada v_1, $e = 25$, $e = 100$

Karena dalam codingan di $RS(255)$ peneliti menggunakan step $s = 5$, maka

$$i = \frac{e}{s}$$

$e = 25 \Rightarrow i = \frac{25}{5} = 5$

$e = 100 \Rightarrow i = \frac{100}{5} = 20$

2. v_2

Polynomial locator $\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

$$\sigma(x) = 1 + \alpha^{244}x + \alpha^{225}x^2$$

$\sigma_0 = 1$

$\sigma_1 = \alpha^{244}$

$\sigma_2 = \alpha^{225}$

$$x = \sigma(\alpha^{-e}) = 0$$

$$\sigma(\alpha^{-e}) = 1 + \alpha^{244}\alpha^{-e} + \alpha^{225}\alpha^{-2e}$$

Untuk setiap $e = 0,1,2,\ldots,254$

$\sigma(\alpha^{-e}) \quad = 1 + \alpha^{244}\alpha^{-e} + \alpha^{225}\alpha^{-2e}$

$\sigma(\alpha^{-50}) = 1 + \alpha^{244}\alpha^{-50} + \alpha^{225}\alpha^{-2\cdot 50}$

$\sigma(\alpha^{-50}) = 1 + \alpha^{244-50} + \alpha^{225-100}$

$\sigma(\alpha^{-50}) = 1 + \alpha^{194} + \alpha^{125}$

$\sigma(\alpha^{-50}) = 0$

$\sigma(\alpha^{-e}) \quad = 1 + \alpha^{244}\alpha^{-e} + \alpha^{225}\alpha^{-2e}$

$\sigma(\alpha^{-175}) = 1 + \alpha^{244}\alpha^{-175} + \alpha^{225}\alpha^{-2\cdot175}$

$\sigma(\alpha^{-175}) = 1 + \alpha^{244-175} + \alpha^{225-350}$

$\sigma(\alpha^{-175}) = 1 + \alpha^{69} + \alpha^{130}$

$\sigma(\alpha^{-175}) = 0$

Jadi lokasi kesalahan pada v_1, $e = 50$, $e = 175$

Karena dalam codingan di $RS(255)$ peneliti menggunakan step $s = 5$, maka

$$i = \frac{e}{s}$$

$e = 50 \Rightarrow i = \frac{50}{5} = 10$

$e = 175 \Rightarrow i = \frac{175}{5} = 35$

4.4.4 Menentukan Akar Polinomial Evaluasi Kesalahan (*Error Evaluator Polynomial*)

1. v_1

$$\Omega(x) = \sigma(x) \cdot S(x) mod x^{2t}$$

$$\sigma(x) = 1 + \alpha^{224}x + \alpha^{125}x^2$$

$s_0 = \alpha^{236}$, $s_1 = \alpha^{163}$, $s_2 = \alpha^{49}$, $s_3 = \alpha^{51}$, $\sigma_1 = \alpha^{224}$, $\sigma_2 = \alpha^{125}$

$\Omega_0 = s_0$

$= \alpha^{236}$

$$\Omega_1 = s_1 \oplus (s_0 \cdot \sigma_1)$$

$$= \alpha^{163} \oplus (\alpha^{236} \cdot \alpha^{224})$$

$$= \alpha^{163} \oplus \alpha^{205} = \alpha^{183}$$

$$\Omega_2 = s_2 \oplus (s_1 \cdot \sigma_1) \oplus (s_0 \cdot \sigma_2)$$

$$= \alpha^{49} \oplus (\alpha^{163} \cdot \alpha^{224}) \oplus (\alpha^{236} \cdot \alpha^{125})$$

$$= \alpha^{49} \oplus \alpha^{132} \oplus \alpha^{106} = 0$$

$$\Omega_3 = s_3 \oplus (s_2 \cdot \sigma_1) \oplus (s_1 \cdot \sigma_2)$$

$$= \alpha^{51} \oplus (\alpha^{49} \cdot \alpha^{224}) \oplus (\alpha^{163} \cdot \alpha^{125})$$

$$= \alpha^{51} \oplus \alpha^{18} \oplus \alpha^{33} = 0$$

$$\Omega(x) = \alpha^{236} + \alpha^{183}x + 0x^2 + 0x^3$$

2. v_2

$$\sigma(x) = 1 + \alpha^{244}x + \alpha^{225}x^2$$

$$\Omega(x) = \sigma(x) \cdot S(x) mod x^{2t}$$

$s_0 = \alpha^{166}, s_1 = \alpha^{138}, s_2 = \alpha^{247}, s_3 = \alpha^{248}, \sigma_1 = \alpha^{244}, \sigma_2 = \alpha^{225}$

$$\Omega_0 = s_0$$

$$= \alpha^{166}$$

$$\Omega_1 = s_1 \oplus (s_0 \cdot \sigma_1)$$

$$= \alpha^{138} \oplus (\alpha^{166} \cdot \alpha^{244})$$

$$= \alpha^{138} \oplus \alpha^{155} = \alpha^{206}$$

$$\Omega_2 = s_2 \oplus (s_1 \cdot \sigma_1) \oplus (s_0 \cdot \sigma_2)$$

$$= \alpha^{247} \oplus (\alpha^{138} \cdot \alpha^{244}) \oplus (\alpha^{166} \cdot \alpha^{225})$$

$$= \alpha^{247} \oplus \alpha^{127} \oplus \alpha^{136} = 0$$

$$\Omega_3 = s_3 \oplus (s_2 \cdot \sigma_1) \oplus (s_1 \cdot \sigma_2)$$

$$= \alpha^{248} \oplus (\alpha^{247} \cdot \alpha^{244}) \oplus (\alpha^{138} \cdot \alpha^{225})$$

$= \alpha^{248} \oplus \alpha^{236} \oplus \alpha^{108} = 0$

$$\Omega(x) = \alpha^{166} + \alpha^{206}x + 0x^2 + 0x^3$$

4.4.5 Menghitung Nilai Besar Kesalahan (*Error Magnitude*)

1. v_1

Untuk $e = 25$ (posisi 5)

$$E_j = -\frac{\Omega(\mathrm{X_j^{-1}})}{\sigma'(\mathrm{X_j^{-1}})}$$

$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

$\sigma'(x) = \sigma_1 + 2\sigma_2 x$

$\sigma'(x) = \sigma_1$

$x = \alpha^{-e} = \alpha^{255-e}$

$\Omega(x) = \alpha^{236} + \alpha^{183}x$

Tetapi, karena $2 = 0$ di $GF(2)$, $2\sigma_2 x = 0$. Jadi,

$$E_j = \frac{\Omega(\alpha^{-ej})}{\sigma_1}$$

$$= \frac{\alpha^{236} + \alpha^{183} \cdot \alpha^{255-25}}{\alpha^{224}}$$

$$= \frac{\alpha^{236} + \alpha^{183} \cdot \alpha^{230}}{\alpha^{224}}$$

$$= \frac{\alpha^{236} + \alpha^{158}}{\alpha^{224}} = \frac{\alpha^{115}}{\alpha^{224}} = \alpha^{146}$$

Untuk $e = 100$ (posisi 20)

$$E_j = -\frac{\Omega(\mathrm{X_j^{-1}})}{\sigma'(\mathrm{X_j^{-1}})}$$

$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

$\sigma'(x) = \sigma_1 + 2\sigma_2 x$

$\sigma'(x) = \sigma_1$

$x = \alpha^{-e} = \alpha^{255-e}$

$\Omega(x) = \alpha^{236} + \alpha^{183} x$

Tetapi, karena $2 = 0$ di $GF(2)$, $2\sigma_2 x = 0$. Jadi,

$$E_j = \frac{\Omega(\alpha^{-ej})}{\sigma_1}$$

$$= \frac{\alpha^{236} + \alpha^{183} \cdot \alpha^{255-100}}{\alpha^{224}}$$

$$= \frac{\alpha^{236} + \alpha^{183} \cdot \alpha^{155}}{\alpha^{224}}$$

$$= \frac{\alpha^{236} + \alpha^{83}}{\alpha^{224}} = \frac{\alpha^{202}}{\alpha^{224}} = \alpha^{233}$$

Jadi, pada v_1 nilai *magnitude error* pada posisi [5,20] adalah $[\alpha^{146}, \alpha^{233}]$.

2. v_2

Untuk $e = 50$ (posisi 10)

$$E_j = -\frac{\Omega(\mathrm{X_j^{-1}})}{\sigma'(\mathrm{X_j^{-1}})}$$

$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

$\sigma'(x) = \sigma_1 + 2\sigma_2 x$

$\sigma'(x) = \sigma_1$

$x = \alpha^{-e} = \alpha^{255-e}$

$\Omega(x) = \alpha^{166} + \alpha^{206} x$

Tetapi, karena $2 = 0$ di $GF(2)$, $2\sigma_2 x = 0$. Jadi,

$$E_j = \frac{\Omega(\alpha^{ej})}{\sigma_1}$$

$$= \frac{\alpha^{166} + \alpha^{206} \cdot \alpha^{255-50}}{\alpha^{244}}$$

$$= \frac{\alpha^{166} + \alpha^{206} \cdot \alpha^{205}}{\alpha^{224}}$$

$$= \frac{\alpha^{166} + \alpha^{156}}{\alpha^{244}} = \frac{\alpha^{177}}{\alpha^{244}} = \alpha^{188}$$

Untuk $e = 175$ (posisi 35)

$$E_j = -\frac{\Omega(\mathrm{X_j^{-1}})}{\sigma'(\mathrm{X_j^{-1}})}$$

$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$

$\sigma'(x) = \sigma_1 + 2\sigma_2 x$

$\sigma'(x) = \sigma_1$

$x = \alpha^{-e} = \alpha^{255-e}$

$\Omega(x) = \alpha^{166} + \alpha^{206} x$

Tetapi, karena $2 = 0$ di $GF(2)$, $2\sigma_2 x = 0$. Jadi,

$$E_j = \frac{\Omega(\alpha^{-ej})}{\sigma_1}$$

$$= \frac{\alpha^{166} + \alpha^{206} \cdot \alpha^{255-175}}{\alpha^{244}}$$

$$= \frac{\alpha^{166} + \alpha^{183} \cdot \alpha^{80}}{\alpha^{224}}$$

$$= \frac{\alpha^{166} + \alpha^{31}}{\alpha^{244}} = \frac{\alpha^{175}}{\alpha^{244}} = \alpha^{186}$$

Jadi, pada v_2 nilai *magnitude error* pada posisi $[10,35]$ adalah $[\alpha^{188}, \alpha^{186}]$.

4.4.6 Melakukan Koreksi Kesalahan pada *Codeword*

Proses koreksi dilakukan menggunakan persamaan:

$$C = v - e$$

1. Koreksi pada *Codeword* (C_1)

   Pada *codeword* pertama terdeteksi dua kesalahan (*error*), yaitu masing-masing pada posisi ke$-[5,20]$ dan nilai magnitude *error*-nya $[\alpha^{146}, \alpha^{233}]$. Dengan demikian proses koreksi dilakukan sebagai:

$$C_1 = v_1 - e$$

$C_1 = [\alpha^{240}, \alpha^{35}, \alpha^{39}, \alpha^{120}, \alpha^{153}, \boldsymbol{\alpha^{73}}, \alpha^{227}, \alpha^{151}, \alpha^{233}, \alpha^{198}, \alpha^{85}, \alpha^{213}, \alpha^{44}, \alpha^{104},$
$\alpha^{3}, \alpha^{50}, \alpha^{26}, \alpha^{251}, \alpha^{173}, \alpha^{65}, \boldsymbol{\alpha^{164}}, \alpha^{32}, \alpha^{25}, \alpha^{231}, \alpha^{133}, \alpha^{110}, \alpha^{113}, \alpha^{1}, \alpha^{43}, \alpha^{210},$
$\alpha^{151}, \alpha^{144}, \alpha^{11}, \alpha^{0}, \alpha^{174}, \alpha^{135}, \alpha^{84}, \alpha^{15}, \alpha^{124}, \alpha^{96}, \alpha^{18}, \alpha^{160}, \alpha^{127}, \alpha^{228}, \alpha^{178},$
$\alpha^{22}, \alpha^{216}, \alpha^{181}, \alpha^{52}, \alpha^{40}, \alpha^{196}] -$
$[0x^{50} + 0x^{49} + 0x^{48} + 0x^{47} + 0x^{46} + \boldsymbol{\alpha^{146}x^{45}} + 0x^{44} + 0x^{43} +$
$0x^{42} + 0x^{41} + 0x^{40} + 0x^{39} + 0x^{38} + 0x^{37} + 0x^{36} + 0x^{35} + 0x^{34} +$
$0x^{33} + 0x^{32} + 0x^{31} + \boldsymbol{\alpha^{233}x^{30}} + 0x^{29} + 0x^{28} + 0x^{27} + 0x^{26} + 0x^{25} +$
$0x^{24} + 0x^{23} + 0x^{22} + 0x^{21} + 0x^{20} + 0x^{19} + 0x^{18} + 0x^{17} + 0x^{16} +$
$0x^{15} + 0x^{14} + 0x^{13} + 0x^{12} + 0x^{11} + 0x^{10} + 0x^{9} + 0x^{8} + 0x^{7} + 0x^{6} +$
$0x^{5} + 0x^{4} + 0x^{3} + 0x^{2} + 0x^{1} + 0]$
$= [\alpha^{240}, \alpha^{35}, \alpha^{39}, \alpha^{120}, \alpha^{153}, \boldsymbol{\alpha^{54}}, \alpha^{227}, \alpha^{151}, \alpha^{233}, \alpha^{198}, \alpha^{85}, \alpha^{213}, \alpha^{44}, \alpha^{104},$
$\alpha^{3}, \alpha^{50}, \alpha^{26}, \alpha^{251}, \alpha^{173}, \alpha^{65}, \boldsymbol{\alpha^{39}}, \alpha^{32}, \alpha^{25}, \alpha^{231}, \alpha^{133}, \alpha^{110}, \alpha^{113}, \alpha^{1}, \alpha^{43}, \alpha^{210},$
$\alpha^{151}, \alpha^{144}, \alpha^{11}, \alpha^{0}, \alpha^{174}, \alpha^{135}, \alpha^{84}, \alpha^{15}, \alpha^{124}, \alpha^{96}, \alpha^{18}, \alpha^{160}, \alpha^{127}, \alpha^{228}, \alpha^{178},$

$\alpha^{22}, \alpha^{216}, \alpha^{181}, \alpha^{52}, \alpha^{40}, \alpha^{196}]$

Hasil ini menunjukkan bahwa kedua error pada posisi $[5,20]$ telah berhasil diperbaiki, sehingga nilai (C_1) kembali sesuai dengan hasil *encoding* awal.

2. Koreksi pada *Codeword* (C_2)

Pada *codeword* kedua terdeteksi dua kesalahan (*error*), yaitu masing-masing pada posisi ke$-[10,35]$ dan nilai magnitude *error*-nya $[\alpha^{188}, \alpha^{186}]$. Dengan demikian proses koreksi dilakukan sebagai:

$$C_2 = v_2 - e$$

$C_2 = [\alpha^{40}, \alpha^{243}, \alpha^{86}, \alpha^{119}, \alpha^{46}, \alpha^{75}, \alpha^{106}, \alpha^{116}, \alpha^{203}, \alpha^{91}, \boldsymbol{\alpha^{196}}, \alpha^{21}, \alpha^{175}, \alpha^{8}, \alpha^{112},$
$\alpha^{21}, \alpha^{27}, \alpha^{231}, \alpha^{241}, \alpha^{22}, \alpha^{205}, \alpha^{216}, \alpha^{22}, \alpha^{84}, \alpha^{101}, \alpha^{246}, \alpha^{67}, \alpha^{30}, \alpha^{184}, \alpha^{96}, \alpha^{28},$
$\alpha^{226}, \alpha^{219}, \alpha^{25}, \alpha^{118}, \boldsymbol{\alpha^{172}}, \alpha^{251}, \alpha^{182}, \alpha^{49}, \alpha^{181}, \alpha^{203}, \alpha^{46}, \alpha^{49}, \alpha^{162}, \alpha^{41}, \alpha^{113}, \alpha^{63},$
$\alpha^{239}, \alpha^{72}, \alpha^{226}] +$

$[0x^{50} + 0x^{49} + 0x^{48} + 0x^{47} + 0x^{46} + 0x^{45} + 0x^{44} + 0x^{43} + 0x^{42} +$
$0x^{41} + \boldsymbol{\alpha^{188}x^{40}} + 0x^{39} + 0x^{38} + 0x^{37} + 0x^{36} + 0x^{35} + 0x^{34} + 0x^{33} +$
$0x^{32} + 0x^{31} + 0x^{30} + 0x^{29} + 0x^{28} + 0x^{27} + 0x^{26} + 0x^{25} + 0x^{24} +$
$0x^{23} + 0x^{22} + 0x^{21} + 0x^{20} + 0x^{19} + 0x^{18} + 0x^{17} + 0x^{16} + \boldsymbol{\alpha^{186}x^{15}} +$
$0x^{14} + 0x^{13} + 0x^{12} + 0x^{11} + 0x^{10} + 0x^{9} + 0x^{8} + 0x^{7} + 0x^{6} + 0x^{5} +$
$0x^{4} + 0x^{3} + 0x^{2} + 0x^{1} + 0] =$

$[\alpha^{40}, \alpha^{243}, \alpha^{86}, \alpha^{119}, \alpha^{46}, \alpha^{75}, \alpha^{106}, \alpha^{116}, \alpha^{203}, \alpha^{91}, \boldsymbol{\alpha^{133}}, \alpha^{21}, \alpha^{175}, \alpha^{8}, \alpha^{112},$
$\alpha^{21}, \alpha^{27}, \alpha^{231}, \alpha^{241}, \alpha^{22}, \alpha^{205}, \alpha^{216}, \alpha^{22}, \alpha^{84}, \alpha^{101}, \alpha^{246}, \alpha^{67}, \alpha^{30}, \alpha^{184}, \alpha^{96}, \alpha^{28},$
$\alpha^{226}, \alpha^{219}, \alpha^{25}, \alpha^{118}, \boldsymbol{\alpha^{141}}, \alpha^{251}, \alpha^{182}, \alpha^{49}, \alpha^{181}, \alpha^{203}, \alpha^{46}, \alpha^{49}, \alpha^{162}, \alpha^{41}, \alpha^{113}, \alpha^{63},$
$\alpha^{239}, \alpha^{72}, \alpha^{226}]$

Hasil ini menunjukkan bahwa kedua *error* pada posisi $[10,35]$ telah berhasil diperbaiki, sehingga nilai (C_2) kembali sesuai dengan hasil *encoding* awal.

4.4.7 Proses *Decoding* untuk Mengembalikan *Codeword* yang Telah Dikoreksi dengan Pesan Asli

Setelah *codeword* terkoreksi maka perlu mengembalikan *codeword* ke bentuk pesan aslinya dengan mengambil k simbol pertama menggunakan SageMath maka di peroleh kembali,

$$C_1 = [\alpha^{26}, \alpha^{226}, \alpha^{26}, \alpha^{33}, \alpha^{26}, \alpha^{48}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{154},$$

$$\alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{102}, \alpha^{26}, \alpha^{48}, 0, \alpha^{5}, \alpha^{26}, \alpha^{221}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{154}, \alpha^{26}, \alpha^{102}$$

$$\alpha^{26}, \alpha^{37}, \alpha^{26}, \alpha^{253}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{39}, \alpha^{26}, \alpha^{154}, \alpha^{26}]$$

$$C_1 = \begin{bmatrix} [00000110][01001000][00000110][00100111][00000110][01000110] \\ [00000110][00100111][00000000][00100000][00000110][01000100] \\ [00000110][00101100][00000110][00111001][00000110][01000100] \\ [00000110][01001000][00000110][01000110][00000000][00100000] \\ [00000110][01000101][00000110][00100111][00000000][00100000] \\ [00000110][00111001][00000110][01000100][00000110][01001010] \\ [00000110][01000111][00000110][00100111][00000000][00100000] \\ [00000110][00110101][00000110][00111001][00000110] \end{bmatrix}$$

$$C_2 = [\alpha^{37}, \alpha^{26}, \alpha^{69}, \alpha^{26}, \alpha^{33}, 0, \alpha^{5}, \alpha^{26}, \alpha^{240}, \alpha^{26}, \alpha^{181}, \alpha^{26}, \alpha^{194}, \alpha^{26}, \alpha^{33}, 0,0,$$

$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

$$C_2 = \begin{bmatrix} [01001010][00000110][00101111][00000110][00100111][00000000] \\ [00100000][00000110][00101100][00000110][00110001][00000110] \\ [00110010][00000110][00100111][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000][00000000] \\ [00000000][00000000][00000000][00000000][00000000] \end{bmatrix}$$

Tahap terakhir dalam proses ini adalah membagi deretan bit tersebut menjadi beberapa blok dengan panjang masing-masing 8-bit. Proses ini bertujuan untuk mengkonversi setiap blok biner menjadi karakter yang sesuai berdasarkan tabel *Unicode*. Setelah dilakukan pembagian dan konversi, diperoleh karakter karakter sebagai berikut:

Tabel 4.3 Hasil *Decoding* Kembali ke Pesan Asli

Huruf	Biner-16 Bit	Huruf	Biner-16 Bit
و	[0000011001001000]	ل	[0000011001000100]
ا	[0000011000100111]	ي	[0000011001001010]
ن	[0000011001000110]	ه	[0000011001000111]
ا	[0000011000100111]	ا	[0000011000100111]
ـ	[0000000000100000]	ـ	[0000000000100000]
ل	[0000011001000100]	ص	[0000011000110101]
ج	[0000011000101100]	ع	[0000011000111001]
ع	[0000011000111001]	ي	[0000011001001010]
ل	[0000011001000100]	د	[0000011000101111]
و	[0000011001001000]	ا	[0000011000100111]
ن	[0000011001000110]	ـ	[0000000000100000]
ـ	[0000000000100000]	ج	[0000011000101100]
م	[0000011001000101]	ر	[0000011000110001]
ا	[0000011000100111]	ز	[0000011000110010]
ـ	[0000000000100000]	ا	[0000011000100111]
ع	[0000011000111001]		

Dengan demikian, hasil akhir dari proses *decoding* berhasil memulihkan pesan asli yang dikirimkan, yakni lafadz "وانا لجعلون ما عليها صعيدا جرزا" secara benar dan tanpa kesalahan.

4.5 Analisis Hasil dengan Beberapa Paremeter Kode Reed-Solomon

Pada penelitian ini, kode Reed–Solomon diterapkan pada sepuluh ayat Al-Qur'an dengan panjang kode tetap $n = 51$. Jumlah kesalahan yang disisipkan pada tiap ayat divariasikan antara satu hingga lima simbol sehingga parameter kemampuan koreksi t menyesuaikan, dengan nilai $k = n - 2t$ dihitung otomatis oleh sistem. Seluruh komputasi dilakukan pada lapangan hingga $GF(2^8)$ menggunakan SageMath. Analisis kinerja difokuskan pada empat aspek utama:

keberhasilan deteksi, keberhasilan koreksi, konsistensi parameter, dan stabilitas *decoding* ketika jumlah *error* meningkat.

Tabel 4.4 Hasil Analisis Parameter Kode RS Sepuluh Ayat Al-Qur'an 1 Kesalahan

No	Surah:Ayat	Banyak Simbol	$RS(n,k,t)$	Posisi Error	Hasil Deteksi & Koreksi	Keterangan
1	18:48	31	(51,49,1)	$C_1 = 5, C_2 = 10$	1 *error*	Sesuai
2	2:2	33	(51,49,1)	$C_1 = 2, C_2 = 7$	1 *error*	Sesuai
3	4:48	39	(51,49,1)	$C_1 = 3, C_2 = 5$	1 *error*	Sesuai
4	8:3	39	(51,49,1)	$C_1 = 1, C_2 = 8$	1 *error*	Sesuai
5	79:15	17	(51,49,1)	$C_1 = 4, C_2 = 2$	1 *error*	Sesuai
6	15:55	38	(51,49,1)	$C_1 = 6, C_2 = 14$	1 *error*	Sesuai
7	23:49	35	(51,49,1)	$C_1 = 0, C_2 = 20$	1 *error*	Sesuai
8	35:16	28	(51,49,1)	$C_1 = 9, C_2 = 3$	1 *error*	Sesuai
9	68:33	46	(51,49,1)	$C_1 = 12, C_2 = 6$	1 *error*	Sesuai
10	89:5	21	(51,49,1)	$C_1 = 2, C_2 = 5$	1 *error*	Sesuai

Berdasarkan Tabel 4.4 yang memuat sepuluh ayat uji pada konfigurasi $t = 1$ pada pengujian ini memakan waktu sekitar 5 detik, seluruh ayat memiliki parameter $(51, 49, 1)$. Konfigurasi ini memungkinkan sistem mendeteksi serta mengoreksi tepat satu kesalahan simbol pada setiap *codeword*. Data pada tabel menunjukkan konsistensi penuh: setiap ayat mengalami satu *error* selama transmisi, dan seluruh *error* berhasil dikoreksi sebagaimana ditunjukkan oleh kecocokan antara kolom "Hasil Deteksi", "Hasil Koreksi", dan keterangan "Sesuai".

Proses *decoding* pada $t = 1$ bersifat paling sederhana karena hanya memerlukan dua sindrom awal, yaitu $S_1 = v(\alpha)$, dan $S_2 = v(\alpha^2)$, dengan $v(x)$ merupakan *codeword* yang diterima. Kedua sindrom tersebut selanjutnya digunakan untuk menyusun polinomial *error locator*.

$$\Lambda(x) = 1 + \lambda_1 x, \qquad \lambda_1 = \frac{S_1}{S_2},$$

Pencarian akar polinomial ini hanya menghasilkan satu posisi kesalahan, dan magnitudo *error* dihitung menggunakan bentuk dasar rumus Forney. Akibatnya,

decoding berlangsung cepat dan deterministik. Seluruh *codeword* berhasil dipulihkan ke bentuk valid.

Tabel 4.5 Hasil Analisis Parameter Kode RS Sepuluh Ayat Al-Qur'an 2 Kesalahan

No	Surah:Ayat	Banyak Simbol	RS (n, k, t)	Posisi Error	Hasil Deteksi & Koreksi	Keterangan
1	18:48	31	(51,47,2)	$C_1 = 5{,}20$ $C_2 = 10{,}35$	2 *error*	Sesuai
2	2:2	33	(51,47,2)	$C_1 = 2{,}14$ $C_2 = 7{,}30$	2 *error*	Sesuai
3	4:48	39	(51,47,2)	$C_1 = 3{,}15$ $C_2 = 5{,}22$	2 *error*	Sesuai
4	8:3	39	(51,47,2)	$C_1 = 1{,}37$ $C_2 = 8{,}3$	2 *error*	Sesuai
5	79:15	17	(51,47,2)	$C_1 = 4{,}18$ $C_2 = 2{,}28$	2 *error*	Sesuai
6	15:55	38	(51,47,2)	$C_1 = 6{,}11$ $C_2 = 14{,}26$	2 *error*	Sesuai
7	23:49	35	(51,47,2)	$C_1 = 0{,}16$ $C_2 = 20{,}40$	2 *error*	Sesuai
8	35:16	28	(51,47,2)	$C_1 = 9{,}21$ $C_2 = 3{,}17$	2 *error*	Sesuai
9	68:33	46	(51,47,2)	$C_1 = 12{,}13$ $C_2 = 6{,}48$	2 *error*	Sesuai
10	89:5	21	(51,47,2)	$C_1 = 2{,}30$ $C_2 = 5{,}25$	2 *error*	Sesuai

Berdasarkan Tabel 4.5, pada pengujian ini memakan waktu sekitar 6 detik dengan konfigurasi $t = 2$ menggunakan parameter $(51, 47, 2)$ **,** sehingga kode Reed–Solomon dapat mengoreksi dua kesalahan simbol dalam satu *codeword*. Proses *decoding* dimulai dengan menghitung empat sindrom awal, yaitu S_1, S_2, S_3 , dan S_4. Keempat sindrom ini memberikan informasi yang cukup untuk memastikan bahwa jumlah kesalahan berada dalam batas toleransi sistem.

Secara matematis, dua kesalahan menghasilkan polinomial *error locator* berderajat dua,

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2,$$

yang dihitung melalui algoritma Berlekamp–Massey. Akar-akar polinomial tersebut menentukan dua posisi simbol yang rusak melalui pengujian $\Lambda(\alpha^{-i}) = 0$. Setelah posisi kesalahan ditemukan, magnitudo masing-masing kesalahan dihitung

menggunakan rumus Forney. Pada kasus dua *error*, perhitungan melibatkan turunan formal $\Lambda'(x)$ serta evaluasi sindrom dalam $GF\ (2^8)$. Koreksi kemudian diberikan dengan mengurangi magnitudo kesalahan dari dua simbol yang rusak, sehingga *codeword* kembali ke bentuk yang valid.

Dengan demikian, meskipun kompleksitas *decoding* meningkat akibat jumlah sindrom yang lebih banyak dan polinomial berderajat lebih tinggi, konfigurasi $t = 2$ tetap mampu memulihkan pesan selama jumlah kesalahan tidak melebihi dua simbol.

Berdasarkan Tabel 4.6, pada pengujian ini memakan waktu sekitar 5 detik dengan peningkatan kemampuan koreksi $t = 3$ dan parameter $(51, 45, 3)$ memungkinkan sistem menangani hingga tiga kesalahan simbol dalam satu *codeword*. Proses *decoding* dimulai dengan perhitungan enam sindrom pertama $(S_1 - S_6)$ yang memberikan informasi lengkap mengenai struktur kerusakan.

Tabel 4.6 Hasil Analisis Parameter Kode RS Sepuluh Ayat Al-Qur'an 3 Kesalahan

No	Surah:Ayat	Banyak Simbol	$RS\ (n, k, t)$	Posisi Error	Hasil Deteksi & Koreksi	Keterangan
1	18:48	31	(51,45,3)	$C_1 = 5{,}20{,}34$ $C_2 = 10{,}35{,}21$	3 *error*	Sesuai
2	2:2	33	(51,45,3)	$C_1 = 2{,}14{,}17$ $C_2 = 7{,}30{,}34$	3 *error*	Sesuai
3	4:48	39	(51,45,3)	$C_1 = 3{,}15{,}30$ $C_2 = 5{,}22{,}47$	3 *error*	Sesuai
4	8:3	39	(51,45,3)	$C_1 = 1{,}37{,}30$ $C_2 = 8{,}45{,}45$	3 *error*	Sesuai
5	79:15	17	(51,45,3)	$C_1 = 4{,}18{,}7$ $C_2 = 2{,}28{,}30$	3 *error*	Sesuai
6	15:55	38	(51,45,3)	$C_1 = 6{,}11{,}24$ $C_2 = 14{,}31{,}47$	3 *error*	Sesuai
7	23:49	35	(51,45,3)	$C_1 = 0{,}2{,}47$ $C_2 = 20{,}40{,}32$	3 *error*	Sesuai
8	35:16	28	(51,45,3)	$C_1 = 9{,}21{,}49$ $C_2 = 3{,}17{,}45$	3 *error*	Sesuai
9	68:33	46	(51,45,3)	$C_1 = 12{,}13{,}14$ $C_2 = 6{,}22{,}20$	3 *error*	Sesuai
10	89:5	21	(51,45,3)	$C_1 = 2{,}30{,}48$ $C_2 = 5{,}25{,}24$	3 *error*	Sesuai

Algoritma Berlekamp–Massey menghasilkan polinomial *error locator* $\Lambda(x)$, berderajat tiga,

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3.$$

Derajat dari polinomial ini bersifat penting, karena mencerminkan jumlah *error* yang terdeteksi. Jika derajatnya tepat tiga, maka sistem secara matematis konsisten bahwa terdapat tiga simbol yang rusak. Setelah posisi *error* diketahui, tahap berikutnya adalah perhitungan magnitudo kesalahan melalui rumus Forney yang telah diperluas untuk kasus lebih dari dua kerusakan. Rumus Forney untuk derajat tiga melibatkan evaluasi turunan polinomial locator serta polinomial evaluator $\Omega(x)$.

Ketika ketiga simbol telah dikoreksi, *codeword* kembali memenuhi seluruh persamaan paritas kode Reed–Solomon. Dengan demikian, pesan asli dapat dipulihkan tanpa kehilangan informasi apa pun. Kasus $t = 3$ memperlihatkan bahwa walaupun beban komputasi meningkat, kode Reed–Solomon tetap dapat bekerja secara optimal untuk koreksi kesalahan multi-simbol selama jumlah kesalahan masih berada di bawah kapasitas koreksi maksimum.

Berdasarkan Tabel 4.7, pada pengujian ini memakan waktu sekitar 5 detik dengan konfigurasi $t = 4$ dan parameter $(51, 43, 4)$ menunjukkan bahwa setiap *codeword* dapat menampung hingga empat kesalahan simbol. Hasil uji memperlihatkan bahwa seluruh ayat yang mengandung empat *error* berhasil dipulihkan secara tepat oleh *decoder*. Hal ini dibuktikan melalui kesesuaian antara jumlah kesalahan yang terdeteksi, kesalahan yang dikoreksi, serta hasil akhir yang kembali identik dengan pesan asli. Dari sisi proses, *decoding* memerlukan delapan sindrom $(S_1, \ldots, S_8)$ dan menghasilkan polinomial *error locator* berderajat empat,

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3 + \lambda_4 x^4.$$

Tabel 4.7 Hasil Analisis Parameter Kode RS Sepuluh Ayat Al-Qur'an 4 Kesalahan

No	Surah:Ayat	Banyak Simbol	RS (n, k, t)	Posisi Error	Hasil Deteksi & Koreksi	Keterangan
1	18:48	31	(51,43,4)	$C_1 = 5,20,22,10$ $C_2 = 10,35,39,27$	4 *error*	Sesuai
2	2:2	33	(51,43,4)	$C_1 = 2,14,15,48$ $C_2 = 7,30,10,10$	4 *error*	Sesuai
3	4:48	39	(51,43,4)	$C_1 = 3,15,30,10$ $C_2 = 5,22,23,50$	4 *error*	Sesuai
4	8:3	39	(51,43,4)	$C_1 = 1,3,44,39$ $C_2 = 8,23,48,45$	4 *error*	Sesuai
5	79:15	17	(51,43,4)	$C_1 = 4,18,1,36$ $C_2 = 2,28,27,26$	4 *error*	Sesuai
6	15:55	38	(51,43,4)	$C_1 = 6,11,24,11$ $C_2 = 14,35,25,17$	4 *error*	Sesuai
7	23:49	35	(51,43,4)	$C_1 = 0,28,14,45$ $C_2 = 20,40,24,11$	4 *error*	Sesuai
8	35:16	28	(51,43,4)	$C_1 = 9,21,19,15$ $C_2 = 3,17,45,32$	4 *error*	Sesuai
9	68:33	46	(51,43,4)	$C_1 = 12,13,14,15$ $C_2 = 6,39,39,25$	4 *error*	Sesuai
10	89:5	21	(51,43,4)	$C_1 = 2,30,48,49$ $C_2 = 5,25,31,20$	4 *error*	Sesuai

Meskipun peningkatan nilai t menyebabkan bertambahnya beban komputasi terutama pada tahap pencarian akar dan perhitungan magnitudo kesalahan seluruh langkah tetap berjalan stabil dan konsisten. Keberhasilan pemulihan seluruh *codeword* pada konfigurasi ini menunjukkan bahwa sistem masih berada dalam batas kapasitas koreksi maksimum dan seluruh mekanisme *decoding* bekerja secara deterministik.

Dengan demikian, pada $t = 4$ algoritma tetap mampu melakukan deteksi serta koreksi multi-simbol dengan tingkat akurasi penuh, meskipun kompleksitas matematis meningkat secara signifikan dibandingkan konfigurasi t yang lebih rendah.

Tabel 4.8 Hasil Analisis Parameter Kode RS Sepuluh Ayat Al-Qur'an 5 Kesalahan

No	Surah:Ayat	Banyak Simbol	RS (n, k, t)	Posisi Error	Hasil Deteksi & Koreksi	Keterangan
1	18:48	31	(51,41,5)	$C_1 = 5,20,0,29,37$ $C_2 = 10,35,5,7,47$	5 *error*	Sesuai
2	2:2	33	(51,41,5)	$C_1 = 2,14,14,13,46$ $C_2 = 7,30,37,21,45$	5 *error*	Sesuai
3	4:48	39	(51,41,5)	$C_1 = 3,15,30,9,23$ $C_2 = 5,22,33,35,25$	5 *error*	Sesuai
4	8:3	39	(51,41,5)	$C_1 = 1,32,15,26,24$ $C_2 = 8,28,2,24,49$	5 *error*	Sesuai
5	79:15	17	(51,41,5)	$C_1 = 4,18,32,17,33$ $C_2 = 2,28,22,8,35$	5 *error*	Sesuai
6	15:55	38	(51,41,5)	$C_1 = 6,11,24,28,6$ $C_2 = 14,47,27,35,48$	5 *error*	Sesuai
7	23:49	35	(51,41,5)	$C_1 = 0,3,13,48,39$ $C_2 = 20,40,36,12,31$	5 *error*	Sesuai
8	35:16	28	(51,41,5)	$C_1 = 9,21,39,41,24$ $C_2 = 3,17,45,38,28$	5 *error*	Sesuai
9	68:33	46	(51,41,5)	$C_1 = 12,13,14,15,5$ $C_2 = 6,10,45,21,33$	5 *error*	Sesuai
10	89:5	21	(51,41,5)	$C_1 = 2,30,48,49,50$ $C_2 = 5,25,1,13,50$	5 *error*	Sesuai

Berdasarkan Tabel 4.8, pada pengujian ini memakan waktu sekitar 9 detik dengan konfigurasi $t = 5$ dan parameter $(51, 41, 5)$ menunjukkan bahwa sistem Reed–Solomon beroperasi pada batas maksimum kemampuan koreksi, yakni lima kesalahan simbol per *codeword*. Hasil pengujian memperlihatkan bahwa seluruh *codeword* yang mengalami lima *error* berhasil dipulihkan sepenuhnya oleh proses *decoding*. Hal ini menegaskan bahwa mekanisme deteksi dan koreksi masih bekerja secara stabil meskipun berada pada tingkat kompleksitas tertinggi. Sepuluh sindrom pertama $(S_1 - S_{10})$ dan derajat polinomial *error locator* berderajat lima,

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3 + \lambda_4 x^4 + \lambda_5 x^5.$$

Menunjukkan jika jumlah sindrom dan derajat polinomial locator yang meningkat menambah beban komputasi, namun tidak mengganggu determinisme algoritma: posisi kelima *error* dapat diidentifikasi dengan benar dan koreksi melalui perhitungan magnitudo berlangsung konsisten. Dengan keberhasilan pemulihan seluruh blok uji, konfigurasi $t = 5$ membuktikan bahwa kode Reed–

Solomon mampu mencapai performa optimal hingga kapasitas koreksi maksimalnya. Sistem tetap akurat selama jumlah kesalahan tidak melebihi lima simbol. Apabila *noise* kanal lebih besar dari kapasitas ini, keberhasilan *decoding* tidak lagi terjamin.

4.6 Kajian Hasil Penelitian dalam Perspektif Islam

Penelitian mengenai implementasi kode Reed–Solomon untuk deteksi dan koreksi kesalahan transmisi ayat Al-Qur'an menggunakan pengkodean huruf hijaiyah menunjukkan bahwa teknologi informasi modern dapat menjadi sarana strategis dalam menjaga keaslian teks wahyu, terutama di era digital. Saat ini, ayat-ayat Al-Qur'an tersebar melalui berbagai platform elektronik seperti aplikasi Al-Qur'an, mushaf digital, perangkat lunak pembelajaran, hingga penyimpanan berbasis awan. Proses pengiriman dan penyimpanan digital tersebut membuka peluang terjadinya distorsi data, misalnya hilangnya huruf, tertukarnya karakter, atau kerusakan bit. Contoh nyata dapat ditemukan pada kasus kesalahan penulisan ayat Al-Kahf ayat 8, di mana huruf ع tertukar menjadi ه, atau kesalahan pada Al-Ankabut ayat 45 yang mengalami pengurangan huruf pada lafadz الصّلَوةَ. Kasus-kasus seperti ini menunjukkan bahwa kesalahan penulisan ayat tetap berpotensi terjadi pada media digital.

Selain itu, potensi kesalahan juga dapat muncul dalam konteks digitalisasi teks, misalnya saat proses pemindaian mushaf, konversi huruf hijaiyah ke kode digital, atau saat penyalinan file melalui jaringan. Kesalahan seperti huruf yang tidak terbaca, karakter yang salah terbaca oleh sistem OCR, atau kerusakan data selama proses penyimpanan dan distribusi digital dapat menyebabkan penyimpangan pada

penulisan ayat. Berdasarkan temuan penelitian, algoritma Reed–Solomon terbukti mampu mendeteksi sekaligus mengoreksi bentuk-bentuk kesalahan tersebut sehingga teks ayat dapat direstorasi kembali sesuai naskah aslinya.

Nilai-nilai Islam mengajarkan pentingnya ketelitian (*itqān*) dan amanah dalam setiap pekerjaan, termasuk dalam penulisan dan penjagaan naskah Al-Qur'an. Penerapan algoritma koreksi kesalahan merupakan representasi nyata dari nilai *itqān*, karena sistem mampu melakukan deteksi dan koreksi hingga satu sampai lima kesalahan pada satu ayat, lalu mengembalikannya ke bentuk asli secara akurat. Hal ini selaras dengan hadits:

إِنَّ اللّٰهَ يُحِبُّ إِذَا عَمِلَ أَحَدُكُمْ عَمَلًا أَنْ يُتْقِنَهُ

Artinya: "*Sesungguhnya Allah mencintai seseorang yang ketika bekerja, ia melakukannya dengan sungguh-sungguh dan sempurna*." (HR. al-Baihaqi)

Dalam penelitian ini, ketelitian sistem direpresentasikan melalui kemampuan Reed–Solomon untuk mendeteksi kesalahan hingga lima *error* pada ayat-ayat Al-Qur'an dan mengembalikannya ke bentuk asli dengan akurat. Hal ini mencerminkan praktik *itqān* dalam konteks digital, di mana teknologi digunakan untuk memastikan setiap karakter atau simbol tetap sesuai dengan naskah asli, tanpa distorsi maupun kehilangan informasi. Keberhasilan sistem ini juga berkontribusi terhadap pencapaian tujuan *maqāṣid al-syarī'ah*, khususnya *hifẓ ad-dīn* (penjagaan agama), karena menjaga kemurnian teks Al-Qur'an merupakan bagian dari menjaga syariat dan ajaran agama dari kesalahan penafsiran akibat kelalaian teknis. Implementasi Reed–Solomon memungkinkan transmisi dan penyimpanan teks Al-Qur'an secara digital lebih aman dan andal, sehingga dapat digunakan dalam mushaf digital, aplikasi pembelajaran, dan platform penyimpanan elektronik

dengan risiko kesalahan yang minimal. Selain itu, penerapan algoritma ini selaras dengan prinsip *al-ṣidq* (keaslian) dan *al-dabṭ* (ketepatan) dalam ilmu *qirā'ah* dan rasm mushaf. Reed–Solomon tidak dimaksudkan untuk memodifikasi atau mengubah teks Al-Qur'an, tetapi untuk memulihkan teks yang telah mengalami gangguan transmisi agar kembali ke bentuk aslinya.

Berdasarkan hasil penelitian, implementasi kode Reed–Solomon tidak hanya mengoreksi kesalahan teknis pada transmisi ayat, tetapi juga mendukung tanggung jawab umat Islam dalam menyampaikan Al-Qur'an secara benar. Mekanisme koreksi otomatis yang terbukti mampu mengembalikan ayat ke bentuk asli membuat distribusi ayat melalui media elektronik menjadi lebih aman dan akurat. Prinsip ini terhubung erat dengan hadis Nabi SAW:

Artinya: "*Sebaik-baik kalian adalah yang belajar Al-Qur'an dan mengajarkannya*" (HR. Bukhari No. 5027)

Dalam konteks penelitian ini, penggunaan kode Reed–Solomon tidak hanya dipahami sebagai proses teknis dalam mengoreksi kesalahan data, tetapi juga sebagai bagian dari ikhtiar menjaga amanah penyampaian Al-Qur'an di era digital. Pada masa ketika ayat-ayat Al-Qur'an disebarkan melalui berbagai media elektronik, kebutuhan untuk memastikan bahwa setiap huruf, kata, dan susunan ayat tersampaikan tanpa perubahan menjadi semakin mendesak. Teknologi koreksi kesalahan seperti Reed–Solomon berfungsi sebagai lapisan perlindungan tambahan yang memastikan teks Al-Qur'an tetap akurat meskipun melewati jaringan digital yang rawan mengalami gangguan, seperti hilangnya bit, tergesernya karakter, atau rusaknya sebagian data.

Dengan demikian, implementasi Reed–Solomon dapat dipandang sebagai bentuk modern dari tugas “mengajarkan” dan “menyampaikan” Al-Qur’an secara benar. Jika pada masa klasik ketelitian para *kuttāb* dan *ḥuffāẓ* menjadi kunci menjaga orisinalitas mushaf, maka pada masa digital ketelitian tersebut diwujudkan melalui sistem yang mampu menjaga integritas simbol huruf hijaiyah dalam proses transmisi. Upaya ini sejalan dengan prinsip amanah dan ketelitian (*itqān*) yang diajarkan Islam, bahwa seluruh urusan yang terkait penyampaian wahyu harus dilakukan secara hati-hati, presisi, dan dapat dipertanggungjawabkan. Dengan kode Reed–Solomon, proses distribusi dan penyebaran ayat Al-Qur’an pada generasi digital dapat berlangsung lebih aman, akurat, dan sesuai dengan standar kebenaran yang dituntut oleh syariat.

BAB V
PENUTUP

5.1 Kesimpulan

1. Simulasi proses deteksi kesalahan pada transmisi ayat Al-Qur'an menggunakan kode Reed-Solomon dengan panjang kode tetap $n = 51$ menunjukkan kinerja yang akurat dan konsisten pada seluruh konfigurasi kemampuan koreksi $t = 1$ hingga $t = 5$. Hasil perhitungan sindrom selalu memberikan nilai non-nol ketika terjadi penyisipan *error*, sehingga mekanisme deteksi berjalan efektif untuk seluruh ayat uji. Penyesuaian panjang pesan $k = n - 2t$ pada setiap konfigurasi tidak menimbulkan gangguan terhadap proses deteksi meskipun jumlah simbol ayat berbeda-beda. Hal ini membuktikan bahwa struktur kode Reed–Solomon bersifat stabil, dan dapat diterapkan secara seragam pada berbagai bentuk data ayat Al-Qur'an.
2. Simulasi proses koreksi kesalahan menunjukkan bahwa kode Reed–Solomon mampu memulihkan seluruh *codeword* yang mengalami kerusakan selama jumlah kesalahan tidak melebihi kapasitas koreksi t. Melalui penerapan algoritma Berlekamp–Massey untuk membentuk polinomial *error locator*, pencarian akar menggunakan metode Chien Search, serta perhitungan magnitudo *error* melalui rumus Forney, sistem berhasil menentukan lokasi dan besar kesalahan secara tepat. Pada seluruh pengujian, mulai dari satu hingga lima simbol *error*, semua ayat berhasil direstorasi ke bentuk semula tanpa kehilangan informasi. Hasil ini menegaskan bahwa kode Reed–Solomon tidak hanya efektif tetapi juga sangat reliabel dalam menjaga integritas teks ayat Al-

Qur’an pada proses transmisi digital, bahkan hingga mencapai batas maksimum kemampuan koreksinya.

5.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, beberapa saran dapat diberikan untuk penelitian selanjutnya guna memperluas cakupan implementasi dan memperdalam pemahaman terkait kode Reed-Solomon untuk deteksi dan koreksi kesalahan transmisi ayat Al-Qur’an menggunakan pengkodean huruf hijaiyah, antara lain:

1. Disarankan untuk menguji sistem pada kondisi transmisi yang lebih kompleks, seperti noise acak (*random noise*), *burst error*, atau kanal komunikasi yang menyerupai kondisi nyata.. Hal ini bertujuan untuk mengevaluasi kestabilan kode Reed-Solomon dalam menghadapi kondisi nyata yang lebih beragam.
2. Sistem dapat dikembangkan lebih lanjut dengan mengintegrasikan kode Reed-Solomon dengan teknik pengkodean lain atau metode kriptografi, seperti McEliece atau Reed-Muller, untuk meningkatkan keandalan dan keamanan transmisi teks Al-Qur’an.
3. Disarankan untuk mengaplikasikan sistem ini pada perangkat digital, aplikasi pembelajaran, atau platform penyalinan ayat Al-Qur’an, sehingga dapat membantu meminimalkan kesalahan sekaligus meningkatkan kualitas pembelajaran dan pengiriman/transmisi teks Al-Qur’an secara digital.

DAFTAR PUSTAKA

Alqahtani, Y. (2013). New Approach of Arabic Encryption / Decryption Technique Using Vigenere Cipher on Mod 39. *International Journal of Advanced Research in IT and Engineering*, *2*(12), 1–9.

Bras-amor, M. (2018). A decoding approach to Reed–Solomon codes from their definition. Mathematics, 6(10), 194. https://doi.org/10.3390/math6100194 *A*

Bierbrauer. J. (2019). Introduction to coding theory (2nd ed). In *Sustainability (Switzerland)* (Vol. 11, Issue 1). Universitat Rostock.

Darmadi, D., Imansyah, F., R. R. Y. (2020). Simulasi Eliminasi Noise Dengan Metode Transformasi Wavelet Berbantuan Graphical User Interface (GUI) Matlab. *Jurnal Fakultas Teknik Universitas Tanjungpura Pontianak*, 8.

Dummit, D. S., & Foote, R. M. (2004). Abstract Algebra (3rd ed). Hoboken, NJ: John Wiley & Sons, Inc.

Gallian, J. A. (2021). *Contemporary abstract algebra* (10th ed.). CRC Press. https://doi.org/10.1201/9781003142331

Hadist riwayat Al-Baihaqi. *As-Sunan al-Kubrá* (Vol. 3). Dār al-Kutub al-'Ilmiyyah.

Hadist riwayat Al-Bukhori. *Ṣaḥīḥ al-Bukhārī* (Vol. 7). Dār Ṭawq al-Najāḥ. Hadist No. 5027.

Jamal, R. P., Haryanto, L., & Amir, A. K. (2012). Konstruksi Kode Reed-Solomon sebagai Kode Siklik dengan Polinomial Generator. *Jurnal Matematika, Statistika dan Komputasi*, *14*(1), 100–105.

Jariyah, A., Suwadi & Hendrantoro, G. (2013). Pengkodean Kanal Reed Solomon berbasis FPGA untuk Transmisi Citra pada Satelit Nano. *Jurnal Teknik POMITS*, *2*(1), 51–56.

Kemenag. (2023). *Empat Kali Beredar Ulang Foto Salah Cetak Al-Kahfi: 8, Ini Penjelasan Kemenag*. https://kemenag.go.id/pers-rilis/empat-kali-beredar-ulang-foto-salah-cetak-al-kahfi-8-ini-penjelasan-kemenag-fSlkJ
(Diakses pada 5 September 2024)

Kementrian Agama Republik Indonesia. (2022). *Al-Qur'an dan terjemahannya.*. Lajnah Penthasihan Mushaf Al-Qur'an, Kementrian Agama RI.

Makhomah, R., Santoso, K. A., & Kamsyakawuni, A. (2021). Pengkodean Teks Menggunakan Kombinasi Hill Cipher dan Operasi XOR. *PRISMA, Prosiding Seminar Nasional Matematika*, *4*, 548–552. Universitas Negeri Semarang.

Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC Press.

Nasution, Z. (2020). Metode Pembelajaran Dalam Pengenalan Huruf Hijaiyah. *Jurnal Al-Fatih*, *III*(1), 173–184. http://jurnal.stit-al-ittihadiyahlabura.ac.id/index.php/alfatih/article/view/85

Oktavia, R. E., Utomo, P. H., & Martini, T. S. (2023). Penerapan Kode Reed Solomon Pada Kriptosistem Mceliece. *FIBONACCI: Jurnal Pendidikan Matematika Dan Matematika*, *9*(1), 79. https://doi.org/10.24853/fbc.9.1.79-88

Republika. (2017). *Kesalahan Penulisan Alquran, Penyeleksian Harus Diperluas*. https://khazanah.republika.co.id/berita/oyl5es396/kesalahan-penulisan-alquran-penyeleksian-harus-diperluas

Riyanto, M. Z. (2019). *Pengkodean Huruf Hijaiyah Untuk Deteksi dan Koreksi Kesalahan Penulisan Ayat Al-Qur'an Menggunakan Kode Linear*.

Shihab, M. Q. (2007). *Mukjizat al-Qur''an (Ditinjau dari Aspek Kebahasaan, Isyarat Ilmiah, dan Pemberitaan Gaib*. https://books.google.co.id/books?id=pD5Djck2jeMC&printsec=frontcover&hl=id#v=onepage&q&f=false

Wicker, S. B. (2005). An Introduction to Reed-Solomon Codes. *Gigiena i Sanitariia*, *1*, 30–32.

Widiastuti, N., Lestari, D., & Dhoruri, A. (2016.) Sifat dan karakteristik kode Reed Solomon beserta aplikasinya pada steganography. *Seminar Nasional Matematika dan Pendidikan Matematika UNY 2016* (hlm. 21–26). Yogyakarta: Universitas Negeri Yogyakarta. ISBN 978-602-73403-1-2

LAMPIRAN

Lampiran 1. Representasi Polinomial, Biner, dan Desimal $GF(256)$

```
=== Lapangan GF(2^8): P(x) = x^8 + x^4 + x^3 + x^2 + 1 ===
Elemen primitif α = 0x02

=== Tabel Representasi GF(256) ===
Pangkat  Polinomial                Biner            Desimal  Hex
α^0      1                         00000001 1       0x01
α^1      a                         00000010 2       0x02
α^2      a^2                       00000100 4       0x04
α^3      a^3                       00001000 8       0x08
α^4      a^4                       00010000 16      0x10
α^5      a^5                       00100000 32      0x20
α^6      a^6                       01000000 64      0x40
α^7      a^7                       10000000 128     0x80
α^8      a^4 + a^3 + a^2 + 1       00011101 29      0x1d
α^9      a^5 + a^4 + a^3 + a       00111010 58      0x3a
α^10     a^6 + a^5 + a^4 + a^2     01110100 116     0x74
α^11     a^7 + a^6 + a^5 + a^3     11101000 232     0xe8
α^12     a^7 + a^6 + a^3 + a^2 + 1 11001101 205     0xcd
α^13     a^7 + a^2 + a + 1         10000111 135     0x87
α^14     a^4 + a + 1               00010011 19      0x13
α^15     a^5 + a^2 + a             00100110 38      0x26
α^16     a^6 + a^3 + a^2           01001100 76      0x4c
α^17     a^7 + a^4 + a^3           10011000 152     0x98
α^18     a^5 + a^3 + a^2 + 1       00101101 45      0x2d
α^19     a^6 + a^4 + a^3 + a       01011010 90      0x5a
α^20     a^7 + a^5 + a^4 + a^2     10110100 180     0xb4
α^21     a^6 + a^5 + a^4 + a^2 + 1 01110101 117     0x75
α^22     a^7 + a^6 + a^5 + a^3 + a 11101010 234     0xea
α^23     a^7 + a^6 + a^3 + 1       11001001 201     0xc9
α^24     a^7 + a^3 + a^2 + a + 1   10001111 143     0x8f
α^25     a + 1                     00000011 3       0x03
α^26     a^2 + a                   00000110 6       0x06
α^27     a^3 + a^2                 00001100 12      0x0c
α^28     a^4 + a^3                 00011000 24      0x18
α^29     a^5 + a^4                 00110000 48      0x30
α^30     a^6 + a^5                 01100000 96      0x60
α^31     a^7 + a^6                 11000000 192     0xc0
α^32     a^7 + a^4 + a^3 + a^2 + 1 10011101 157     0x9d
α^33     a^5 + a^2 + a + 1         00100111 39      0x27
α^34     a^6 + a^3 + a^2 + a       01001110 78      0x4e
α^35     a^7 + a^4 + a^3 + a^2     10011100 156     0x9c
α^36     a^5 + a^2 + 1             00100101 37      0x25
α^37     a^6 + a^3 + a             01001010 74      0x4a
α^38     a^7 + a^4 + a^2           10010100 148     0x94
α^39     a^5 + a^4 + a^2 + 1       00110101 53      0x35
α^40     a^6 + a^5 + a^3 + a       01101010 106     0x6a
α^41     a^7 + a^6 + a^4 + a^2     11010100 212     0xd4
α^42     a^7 + a^5 + a^4 + a^2 + 1 10110101 181     0xb5
α^43     a^6 + a^5 + a^4 + a^2 + a + 1     01110111 119     0x77
α^44     a^7 + a^6 + a^5 + a^3 + a^2 + a   11101110 238     0xee
α^45     a^7 + a^6 + 1             11000001 193     0xc1
α^46     a^7 + a^4 + a^3 + a^2 + a + 1     10011111 159     0x9f
α^47     a^5 + a + 1               00100011 35      0x23
α^48     a^6 + a^2 + a             01000110 70      0x46
α^49     a^7 + a^3 + a^2           10001100 140     0x8c
α^50     a^2 + 1                   00000101 5       0x05
α^51     a^3 + a                   00001010 10      0x0a
α^52     a^4 + a^2                 00010100 20      0x14
α^53     a^5 + a^3                 00101000 40      0x28
α^54     a^6 + a^4                 01010000 80      0x50
```

α^55	a^7 + a^5	10100000	160	0xa0
α^56	a^6 + a^4 + a^3 + a^2 + 1	01011101	93	0x5d
α^57	a^7 + a^5 + a^4 + a^3 + a	10111010	186	0xba
α^58	a^6 + a^5 + a^3 + 1	01101001	105	0x69
α^59	a^7 + a^6 + a^4 + a	11010010	210	0xd2
α^60	a^7 + a^5 + a^4 + a^3 + 1	10111001	185	0xb9
α^61	a^6 + a^5 + a^3 + a^2 + a + 1	01101111	111	0x6f
α^62	a^7 + a^6 + a^4 + a^3 + a^2 + a	11011110	222	0xde
α^63	a^7 + a^5 + 1	10100001	161	0xa1
α^64	a^6 + a^4 + a^3 + a^2 + a + 1	01011111	95	0x5f
α^65	a^7 + a^5 + a^4 + a^3 + a^2 + a	10111110	190	0xbe
α^66	a^6 + a^5 + 1	01100001	97	0x61
α^67	a^7 + a^6 + a	11000010	194	0xc2
α^68	a^7 + a^4 + a^3 + 1	10011001	153	0x99
α^69	a^5 + a^3 + a^2 + a + 1	00101111	47	0x2f
α^70	a^6 + a^4 + a^3 + a^2 + a	01011110	94	0x5e
α^71	a^7 + a^5 + a^4 + a^3 + a^2	10111100	188	0xbc
α^72	a^6 + a^5 + a^2 + 1	01100101	101	0x65
α^73	a^7 + a^6 + a^3 + a	11001010	202	0xca
α^74	a^7 + a^3 + 1	10001001	137	0x89
α^75	a^3 + a^2 + a + 1	00001111	15	0x0f
α^76	a^4 + a^3 + a^2 + a	00011110	30	0x1e
α^77	a^5 + a^4 + a^3 + a^2	00111100	60	0x3c
α^78	a^6 + a^5 + a^4 + a^3	01111000	120	0x78
α^79	a^7 + a^6 + a^5 + a^4	11110000	240	0xf0
α^80	a^7 + a^6 + a^5 + a^4 + a^3 + a^2 + 1	11111101	253	0xfd
α^81	a^7 + a^6 + a^5 + a^2 + a + 1	11100111	231	0xe7
α^82	a^7 + a^6 + a^4 + a + 1	11010011	211	0xd3
α^83	a^7 + a^5 + a^4 + a^3 + a + 1	10111011	187	0xbb
α^84	a^6 + a^5 + a^3 + a + 1	01101011	107	0x6b
α^85	a^7 + a^6 + a^4 + a^2 + a	11010110	214	0xd6
α^86	a^7 + a^5 + a^4 + 1	10110001	177	0xb1
α^87	a^6 + a^5 + a^4 + a^3 + a^2 + a + 1	01111111	127	0x7f
α^88	a^7 + a^6 + a^5 + a^4 + a^3 + a^2 + a	11111110	254	0xfe
α^89	a^7 + a^6 + a^5 + 1	11100001	225	0xe1
α^90	a^7 + a^6 + a^4 + a^3 + a^2 + a + 1	11011111	223	0xdf
α^91	a^7 + a^5 + a + 1	10100011	163	0xa3
α^92	a^6 + a^4 + a^3 + a + 1	01011011	91	0x5b
α^93	a^7 + a^5 + a^4 + a^2 + a	10110110	182	0xb6
α^94	a^6 + a^5 + a^4 + 1	01110001	113	0x71
α^95	a^7 + a^6 + a^5 + a	11100010	226	0xe2
α^96	a^7 + a^6 + a^4 + a^3 + 1	11011001	217	0xd9
α^97	a^7 + a^5 + a^3 + a^2 + a + 1	10101111	175	0xaf
α^98	a^6 + a + 1	01000011	67	0x43
α^99	a^7 + a^2 + a	10000110	134	0x86
α^100	a^4 + 1	00010001	17	0x11
α^101	a^5 + a	00100010	34	0x22
α^102	a^6 + a^2	01000100	68	0x44
α^103	a^7 + a^3	10001000	136	0x88
α^104	a^3 + a^2 + 1	00001101	13	0x0d
α^105	a^4 + a^3 + a	00011010	26	0x1a
α^106	a^5 + a^4 + a^2	00110100	52	0x34
α^107	a^6 + a^5 + a^3	01101000	104	0x68
α^108	a^7 + a^6 + a^4	11010000	208	0xd0
α^109	a^7 + a^5 + a^4 + a^3 + a^2 + 1	10111101	189	0xbd
α^110	a^6 + a^5 + a^2 + a + 1	01100111	103	0x67
α^111	a^7 + a^6 + a^3 + a^2 + a	11001110	206	0xce
α^112	a^7 + 1	10000001	129	0x81
α^113	a^4 + a^3 + a^2 + a + 1	00011111	31	0x1f
α^114	a^5 + a^4 + a^3 + a^2 + a	00111110	62	0x3e
α^115	a^6 + a^5 + a^4 + a^3 + a^2	01111100	124	0x7c

α^116	a^7 + a^6 + a^5 + a^4 + a^3	11111000	248	0xf8
α^117	a^7 + a^6 + a^5 + a^3 + a^2 + 1	11101101	237	0xed
α^118	a^7 + a^6 + a^2 + a + 1	11000111	199	0xc7
α^119	a^7 + a^4 + a + 1	10010011	147	0x93
α^120	a^5 + a^4 + a^3 + a + 1	00111011	59	0x3b
α^121	a^6 + a^5 + a^4 + a^2 + a	01110110	118	0x76
α^122	a^7 + a^6 + a^5 + a^3 + a^2	11101100	236	0xec
α^123	a^7 + a^6 + a^2 + 1	11000101	197	0xc5
α^124	a^7 + a^4 + a^2 + a + 1	10010111	151	0x97
α^125	a^5 + a^4 + a + 1	00110011	51	0x33
α^126	a^6 + a^5 + a^2 + a	01100110	102	0x66
α^127	a^7 + a^6 + a^3 + a^2	11001100	204	0xcc
α^128	a^7 + a^2 + 1	10000101	133	0x85
α^129	a^4 + a^2 + a + 1	00010111	23	0x17
α^130	a^5 + a^3 + a^2 + a	00101110	46	0x2e
α^131	a^6 + a^4 + a^3 + a^2	01011100	92	0x5c
α^132	a^7 + a^5 + a^4 + a^3	10111000	184	0xb8
α^133	a^6 + a^5 + a^3 + a^2 + 1	01101101	109	0x6d
α^134	a^7 + a^6 + a^4 + a^3 + a	11011010	218	0xda
α^135	a^7 + a^5 + a^3 + 1	10101001	169	0xa9
α^136	a^6 + a^3 + a^2 + a + 1	01001111	79	0x4f
α^137	a^7 + a^4 + a^3 + a^2 + a	10011110	158	0x9e
α^138	a^5 + 1	00100001	33	0x21
α^139	a^6 + a	01000010	66	0x42
α^140	a^7 + a^2	10000100	132	0x84
α^141	a^4 + a^2 + 1	00010101	21	0x15
α^142	a^5 + a^3 + a	00101010	42	0x2a
α^143	a^6 + a^4 + a^2	01010100	84	0x54
α^144	a^7 + a^5 + a^3	10101000	168	0xa8
α^145	a^6 + a^3 + a^2 + 1	01001101	77	0x4d
α^146	a^7 + a^4 + a^3 + a	10011010	154	0x9a
α^147	a^5 + a^3 + 1	00101001	41	0x29
α^148	a^6 + a^4 + a	01010010	82	0x52
α^149	a^7 + a^5 + a^2	10100100	164	0xa4
α^150	a^6 + a^4 + a^2 + 1	01010101	85	0x55
α^151	a^7 + a^5 + a^3 + a	10101010	170	0xaa
α^152	a^6 + a^3 + 1	01001001	73	0x49
α^153	a^7 + a^4 + a	10010010	146	0x92
α^154	a^5 + a^4 + a^3 + 1	00111001	57	0x39
α^155	a^6 + a^5 + a^4 + a	01110010	114	0x72
α^156	a^7 + a^6 + a^5 + a^2	11100100	228	0xe4
α^157	a^7 + a^6 + a^4 + a^2 + 1	11010101	213	0xd5
α^158	a^7 + a^5 + a^4 + a^2 + a + 1	10110111	183	0xb7
α^159	a^6 + a^5 + a^4 + a + 1	01110011	115	0x73
α^160	a^7 + a^6 + a^5 + a^2 + a	11100110	230	0xe6
α^161	a^7 + a^6 + a^4 + 1	11010001	209	0xd1
α^162	a^7 + a^5 + a^4 + a^3 + a^2 + a + 1	10111111	191	0xbf
α^163	a^6 + a^5 + a + 1	01100011	99	0x63
α^164	a^7 + a^6 + a^2 + a	11000110	198	0xc6
α^165	a^7 + a^4 + 1	10010001	145	0x91
α^166	a^5 + a^4 + a^3 + a^2 + a + 1	00111111	63	0x3f
α^167	a^6 + a^5 + a^4 + a^3 + a^2 + a	01111110	126	0x7e
α^168	a^7 + a^6 + a^5 + a^4 + a^3 + a^2	11111100	252	0xfc
α^169	a^7 + a^6 + a^5 + a^2 + 1	11100101	229	0xe5
α^170	a^7 + a^6 + a^4 + a^2 + a + 1	11010111	215	0xd7
α^171	a^7 + a^5 + a^4 + a + 1	10110011	179	0xb3
α^172	a^6 + a^5 + a^4 + a^3 + a + 1	01111011	123	0x7b
α^173	a^7 + a^6 + a^5 + a^4 + a^2 + a	11110110	246	0xf6
α^174	a^7 + a^6 + a^5 + a^4 + 1	11110001	241	0xf1
α^175	a^7 + a^6 + a^5 + a^4 + a^3 + a^2 + a + 1	11111111	255	0xff
α^176	a^7 + a^6 + a^5 + a + 1	11100011	227	0xe3
α^177	a^7 + a^6 + a^4 + a^3 + a + 1	11011011	219	0xdb
α^178	a^7 + a^5 + a^3 + a + 1	10101011	171	0xab

α^179	a^6 + a^3 + a + 1	01001011	75	0x4b
α^180	a^7 + a^4 + a^2 + a	10010110	150	0x96
α^181	a^5 + a^4 + 1	00110001	49	0x31
α^182	a^6 + a^5 + a	01100010	98	0x62
α^183	a^7 + a^6 + a^2	11000100	196	0xc4
α^184	a^7 + a^4 + a^2 + 1	10010101	149	0x95
α^185	a^5 + a^4 + a^2 + a + 1	00110111	55	0x37
α^186	a^6 + a^5 + a^3 + a^2 + a	01101110	110	0x6e
α^187	a^7 + a^6 + a^4 + a^3 + a^2	11011100	220	0xdc
α^188	a^7 + a^5 + a^2 + 1	10100101	165	0xa5
α^189	a^6 + a^4 + a^2 + a + 1	01010111	87	0x57
α^190	a^7 + a^5 + a^3 + a^2 + a	10101110	174	0xae
α^191	a^6 + 1	01000001	65	0x41
α^192	a^7 + a	10000010	130	0x82
α^193	a^4 + a^3 + 1	00011001	25	0x19
α^194	a^5 + a^4 + a	00110010	50	0x32
α^195	a^6 + a^5 + a^2	01100100	100	0x64
α^196	a^7 + a^6 + a^3	11001000	200	0xc8
α^197	a^7 + a^3 + a^2 + 1	10001101	141	0x8d
α^198	a^2 + a + 1	00000111	7	0x07
α^199	a^3 + a^2 + a	00001110	14	0x0e
α^200	a^4 + a^3 + a^2	00011100	28	0x1c
α^201	a^5 + a^4 + a^3	00111000	56	0x38
α^202	a^6 + a^5 + a^4	01110000	112	0x70
α^203	a^7 + a^6 + a^5	11100000	224	0xe0
α^204	a^7 + a^6 + a^4 + a^3 + a^2 + 1	11011101	221	0xdd
α^205	a^7 + a^5 + a^2 + a + 1	10100111	167	0xa7
α^206	a^6 + a^4 + a + 1	01010011	83	0x53
α^207	a^7 + a^5 + a^2 + a	10100110	166	0xa6
α^208	a^6 + a^4 + 1	01010001	81	0x51
α^209	a^7 + a^5 + a	10100010	162	0xa2
α^210	a^6 + a^4 + a^3 + 1	01011001	89	0x59
α^211	a^7 + a^5 + a^4 + a	10110010	178	0xb2
α^212	a^6 + a^5 + a^4 + a^3 + 1	01111001	121	0x79
α^213	a^7 + a^6 + a^5 + a^4 + a	11110010	242	0xf2
α^214	a^7 + a^6 + a^5 + a^4 + a^3 + 1	11111001	249	0xf9
α^215	a^7 + a^6 + a^5 + a^3 + a^2 + a + 1	11101111	239	0xef
α^216	a^7 + a^6 + a + 1	11000011	195	0xc3
α^217	a^7 + a^4 + a^3 + a + 1	10011011	155	0x9b
α^218	a^5 + a^3 + a + 1	00101011	43	0x2b
α^219	a^6 + a^4 + a^2 + a	01010110	86	0x56
α^220	a^7 + a^5 + a^3 + a^2	10101100	172	0xac
α^221	a^6 + a^2 + 1	01000101	69	0x45
α^222	a^7 + a^3 + a	10001010	138	0x8a
α^223	a^3 + 1	00001001	9	0x09
α^224	a^4 + a	00010010	18	0x12
α^225	a^5 + a^2	00100100	36	0x24
α^226	a^6 + a^3	01001000	72	0x48
α^227	a^7 + a^4	10010000	144	0x90
α^228	a^5 + a^4 + a^3 + a^2 + 1	00111101	61	0x3d
α^229	a^6 + a^5 + a^4 + a^3 + a	01111010	122	0x7a
α^230	a^7 + a^6 + a^5 + a^4 + a^2	11110100	244	0xf4
α^231	a^7 + a^6 + a^5 + a^4 + a^2 + 1	11110101	245	0xf5
α^232	a^7 + a^6 + a^5 + a^4 + a^2 + a + 1	11110111	247	0xf7
α^233	a^7 + a^6 + a^5 + a^4 + a + 1	11110011	243	0xf3
α^234	a^7 + a^6 + a^5 + a^4 + a^3 + a + 1	11111011	251	0xfb
α^235	a^7 + a^6 + a^5 + a^3 + a + 1	11101011	235	0xeb
α^236	a^7 + a^6 + a^3 + a + 1	11001011	203	0xcb
α^237	a^7 + a^3 + a + 1	10001011	139	0x8b
α^238	a^3 + a + 1	00001011	11	0x0b
α^239	a^4 + a^2 + a	00010110	22	0x16
α^240	a^5 + a^3 + a^2	00101100	44	0x2c

α^241	a^6 + a^4 + a^3	01011000	88	0x58
α^242	a^7 + a^5 + a^4	10110000	176	0xb0
α^243	a^6 + a^5 + a^4 + a^3 + a^2 + 1	01111101	125	0x7d
α^244	a^7 + a^6 + a^5 + a^4 + a^3 + a	11111010	250	0xfa
α^245	a^7 + a^6 + a^5 + a^3 + 1	11101001	233	0xe9
α^246	a^7 + a^6 + a^3 + a^2 + a + 1	11001111	207	0xcf
α^247	a^7 + a + 1	10000011	131	0x83
α^248	a^4 + a^3 + a + 1	00011011	27	0x1b
α^249	a^5 + a^4 + a^2 + a	00110110	54	0x36
α^250	a^6 + a^5 + a^3 + a^2	01101100	108	0x6c
α^251	a^7 + a^6 + a^4 + a^3	11011000	216	0xd8
α^252	a^7 + a^5 + a^3 + a^2 + 1	10101101	173	0xad
α^253	a^6 + a^2 + a + 1	01000111	71	0x47
α^254	a^7 + a^3 + a^2 + a	10001110	142	0x8e
α^255	1	00000001	1	0x01

Lampiran 2. *Script Code* Sagemath: Proses *Encoding* dan *Decoding*

```
import random

R.<x> = PolynomialRing(GF(2))
# irreducible polynomial: x^8 + x^4 + x^3 + x^2 + 1 (0x11D)
F.<a> = GF(2^8, modulus = x^8 + x^4 + x^3 + x^2 + 1)

print("=== Lapangan GF(2^8): P(x) = x^8 + x^4 + x^3 + x^2 + 1 ===")
print("Elemen primitif α = 0x02\n")

# ====== Helper ======
def poly_to_int(p):
    coeffs = p.coefficients(sparse=False)
    val = 0
    for i, c in enumerate(coeffs):
        if int(c) == 1:
            val += (1 << i)
    return val

def element_to_alpha_power(elem):
    if elem == 0:
        return "0"
    for i in range(255):
        if a**i == elem:
            return f"α^{i}"
    return "?"

# ====== Konversi Arabic -> 16-bit biner per huruf ======
def arabic_to_binary16(text):
    hasil = []
    for i, huruf in enumerate(text, start=1):
        if huruf == " ":
            bits = "0000000000100000"
        else:
            bits = format(ord(huruf), '016b')
        hasil.append((i, huruf, bits))
    return hasil

# ====== Ubah teks -> dua blok k-symbol (8-bit/simbol) dinamically ======
def text_to_two_blocks_for_k(text, k):
    """
    Menghasilkan dua blok masing-masing panjang k (symbol 8-bit).
    Jika data kurang, dipad dengan 0; jika lebih, di-truncate.
    """
    bytes8 = []
    for huruf in text:
        if huruf == " ":
            bits16 = "0000000000100000"
        else:
            bits16 = format(ord(huruf), "016b")
        bytes8.append(bits16[:8])
        bytes8.append(bits16[8:])
    ints = [int(b,2) for b in bytes8]
    # pad agar minimal 2*k simbol
    if len(ints) < 2*k:
        ints += [0]*(2*k - len(ints))
    # truncate jika lebih
    if len(ints) > 2*k:
        ints = ints[:2*k]
    m1 = ints[:k]
    m2 = ints[k:2*k]
    return m1, m2

# ====== konversi int -> GF element (sesuai field) ======
def int_to_GF256(v):
    bits = [int(b) for b in format(v, '08b')]
    p = sum(bits[-(i+1)] * x**i for i in range(8))
    return F(p)

# ====== Syndromes, BM, Chien, Forney (generik) ======
def syndromes_from_vector_sagemath(r, t, step=5, start=0):
    n_local = len(r)
    S = []
```

```
    for j in range(1, 2*t + 1):
        sj = F(0)
        for i in range(n_local):
            sj += r[i] * (a**((start + step*i) * j))
        S.append(sj)
    return S

def berlekamp_massey(S):
    N = len(S)
    C = [F(1)] + [F(0)] * N
    B = [F(1)] + [F(0)] * N
    L = 0; m = 1; b = F(1)
    for n in range(N):
        d = S[n]
        for i in range(1, L+1):
            d += C[i]*S[n-i]
        if d == 0:
            m += 1
        else:
            coef = d / b
            T = C[:]
            for i in range(0, N+1-m):
                C[i+m] -= coef * B[i]
            if 2*L <= n:
                L = n + 1 - L
                B = T
                b = d
                m = 1
            else:
                m += 1
    return C[:L+1], L

def chien_search(sigma_coeffs, n_local, step=5, start=0):
    errors = []
    for i in range(n_local):
        xinv = a**(-(start + step*i))
        val = F(0)
        for j, c in enumerate(sigma_coeffs):
            val += c * (xinv**j)
        if val == 0:
            errors.append(i)
    return errors

def forney_magnitudes(S, sigma_coeffs, error_positions, t, step=5, start=0):
    PR = PolynomialRing(F, 'X')
    X = PR.gen()
    S_poly = PR(0)
    for j, Sj in enumerate(S):
        S_poly += Sj * X**j
    sigma_poly = PR(0)
    for j, cj in enumerate(sigma_coeffs):
        sigma_poly += cj * X**j
    Omega = (S_poly * sigma_poly) % (X**(2*t))
    sigma_prime = sigma_poly.derivative()
    magnitudes = {}
    for pos in error_positions:
        xinv = a**(-(start + step*pos))
        num = Omega(xinv)
        den = sigma_prime(xinv)
        if den == 0:
            magnitudes[pos] = None
        else:
            magnitudes[pos] = - num / den
    return magnitudes

# ====== Fungsi tampil codeword ======
def show_cw(title, cw):
    print(f"\n--- {title} ---")
    print("α^i:", [element_to_alpha_power(c) for c in cw])
    print("dec:", [poly_to_int(c.polynomial()) for c in cw])
    print("hex:", [f"0x{poly_to_int(c.polynomial()):02x}" for c in cw])

# ====== MAIN: proses 10 ayat (t dipaksa 2) ======
n = 51  # tetap
```

```
FORCED_T = 1  # memaksa 2 error per codeword
daftar_ayat = [
    # (teks_arab, ep1_list, ev1_list, ep2_list, ev2_list)
    ("5,20] ,"وانا لجعلون ما عليها صعيدا جرزا], [0x9A,0xF3], [10,35],
[0xA5,0x6E]),
    ("2,14] ,"يسئلونك عن الساعة ايان مرساها], [0xB1,0xC3], [7,30], [0x9F,0x77]),
    ("3,15,30] ,"فاما من اعطى واتقى], [0xB2,0xC4,0x5E], [5,22], [0x88,0xAA]),
    ("1] ,"اقم الصلاة لذكري], [0xA2], [8], [0x5B]),
    ("4,18] ,"هل أتاك حديث موسى], [0x91,0xE1], [2,28], [0x77,0x33]),
    ("6,11,24] ,"ربنا اغفر لنا وارحمنا], [0xAF,0xC8,0x10], [14], [0x99]),
    ("0] ,"انما المؤمنون اخوة], [0x80], [20,40], [0xDE,0xAD]),
    ("9,21] ,"واذا سألك عبادي عني], [0xFE,0x01], [3,17,45], [0x55,0x66,0x77]),
    ("12,13,14,15] ,"فاسألوا اهل الذكر ان كنتم لا تعلمون], [0x11,0x22,0x33,0x44],
[6], [0xAB]),
    ("5:][2,30,48,49,50] ,"قل جاء الحق وزهق الباطل], [0x12,0x34,0x56,0x78,0x9A],
[5,25], [0xDE,0xAD])
]

# Pastikan tepat 10
if len(daftar_ayat) != 10:
    raise ValueError("daftar_ayat harus berisi tepat 10 entri.")

for idx, (text, ep1, ev1, ep2, ev2) in enumerate(daftar_ayat, start=1):
    print("\n" + "="*80)
    print(f"=== Proses Ayat {idx} ===")
    print("="*80)
    print("Teks Arab:", text)

    # ---------------- MODIF: PAKSA 2 ERROR SAJA -----------------
    # ambil hanya 2 elemen jika ada, kalau kosong buat sampai 2 error acak
    ep1 = list(ep1)[:FORCED_T]
    ev1 = list(ev1)[:FORCED_T]
    ep2 = list(ep2)[:FORCED_T]
    ev2 = list(ev2)[:FORCED_T]

    # jika kurang dari FORCED_T, tambahkan posisi/magnitude acak nonzero
    while len(ep1) < FORCED_T:
        pos = random.randint(0, n-1)
        mag = random.randint(1,255)
        ep1.append(pos)
        ev1.append(mag)
    while len(ep2) < FORCED_T:
        pos = random.randint(0, n-1)
        mag = random.randint(1,255)
        ep2.append(pos)
        ev2.append(mag)

    # sekarang t_needed = FORCED_T (2)
    t_needed = FORCED_T
    # hitung k sesuai t_needed
    k_ayat = n - 2 * t_needed  # t=2 -> k=47
    if k_ayat <= 0:
        raise ValueError(f"t_needed={t_needed} terlalu besar -> k_ayat={k_ayat}
<= 0. Kurangi jumlah error.")

    print(f"Memaksa t = {t_needed} (2 error per codeword). Jadi k = {k_ayat}
(simbol pesan per block).")
    # ------------------------------------------------------------

    # 1) tabel 16-bit
    print("\n=== Pesan Arab ke Representasi Biner (16-bit) ===")
    tbl = arabic_to_binary16(text)
    for no, huruf, bits in tbl:
        print(f"{no:2d}\t{huruf}\t{bits}")

    # 2) buat blok k-symbol sesuai k_ayat
    m1_full, m2_full = text_to_two_blocks_for_k(text, k_ayat)
    print(f"\nBlok pesan 1 (panjang asli {len(m1_full)} simbol):\n{m1_full}\n")
    print(f"Blok pesan 2 (panjang asli {len(m2_full)} simbol):\n{m2_full}\n")

    # pad/truncate m1 dan m2 agar panjang == k_ayat (fungsi already does, still
keep)
    def adjust_block_for_k(block, k):
        bl = list(block)
```

```
        if len(bl) < k:
            bl = bl + [0] * (k - len(bl))
            print(f" - Block dipad dari {len(block)} -> {k} simbol (dengan 0).")
        elif len(bl) > k:
            bl = bl[:k]
            print(f" - PERINGATAN: Block dipotong dari {len(block)} -> {k}
simbol (truncate!).")
        return bl

    m1 = adjust_block_for_k(m1_full, k_ayat)
    m2 = adjust_block_for_k(m2_full, k_ayat)

    # 3) bangun RS khusus ayat ini (k berubah)
    print(f"\n=== Bangun RS untuk ayat {idx}: n={n}, k={k_ayat} (t capability =
{(n-k_ayat)//2}) ===")
    C = codes.ReedSolomonCode(F, n, k_ayat)
    G = C.generator_matrix()
    H = C.parity_check_matrix()

    # --- TAMBAHAN: fungsi untuk menampilkan matriks G dan H secara rapi ---
    def pretty_print_matrix_alpha_hex(name, M):
        try:
            rows = M.nrows()
            cols = M.ncols()
            print(f"\n{name} (nrows={rows}, ncols={cols}):")
            for r in range(rows):
                row = M.row(r)
                alpha_row = [element_to_alpha_power(el) for el in row]
                hex_row = []
                for el in row:
                    try:
                        dec = poly_to_int(el.polynomial())
                        hex_row.append(f"0x{dec:02x}")
                    except Exception:
                        # fallback if el not polynomial
                        try:
                            hex_row.append(hex(int(el)))
                        except Exception:
                            hex_row.append(str(el))
                print(f"r{r:02d} | alpha: {alpha_row} | hex: {hex_row}")
        except Exception as e:
            print(f"Gagal tampilkan matriks {name}: {e}")

    # panggil pretty print untuk G dan H
    pretty_print_matrix_alpha_hex("Generator matrix G", G)
    pretty_print_matrix_alpha_hex("Parity-check matrix H", H)

    # 4) encoding (pakai G ayat ini)
    def encode_rs_with_G(msg_bits, G_local):
        # v harus length k_ayat
        v = vector(F, [F(s) for s in msg_bits])
        c = v * G_local
        return list(c)

    def make_message_vector_from_ints(int_list):
        # setiap elemen int_list harus 0..255
        return vector(F, [ int_to_GF256(int(v)) for v in int_list ])

    # contoh penggunaan untuk m1 dan m2:
    msg_vec1 = make_message_vector_from_ints(m1)   # vektor atas F, panjang
k_ayat
    msg_vec2 = make_message_vector_from_ints(m2)

    # sekarang gunakan C.encode (mengharapkan vector atas F, panjang = k)
    cw = C.encode(msg_vec1)
    cw2 = C.encode(msg_vec2)

    print("\n=== Codeword 1 (α^i) ===")
    print([element_to_alpha_power(c) for c in cw])
    print("\n=== Codeword 2 (α^i) ===")
    print([element_to_alpha_power(c) for c in cw2])
    show_cw("cw", cw)
    show_cw("cw2", cw2)
```

```
    # 5) sisip error sesuai input (pos harus < n)
    print("\n=== Simulasi Penambahan Error ===")
    print("Codeword1: posisi", ep1, "nilai(hex)", [hex(v) for v in ev1], " =>
(α^i):", [element_to_alpha_power(int_to_GF256(v)) for v in ev1])
    print("Codeword2: posisi", ep2, "nilai(hex)", [hex(v) for v in ev2], " =>
(α^i):", [element_to_alpha_power(int_to_GF256(v)) for v in ev2])

    def add_errors_to_cw(cw, positions, values):
        corrupted = list(cw)
        for pos, val in zip(positions, values):
            if pos < 0 or pos >= len(corrupted):
                raise IndexError(f"Posisi error {pos} di luar jangkauan
(0..{len(corrupted)-1})")
            corrupted[pos] += int_to_GF256(val)
        return corrupted

    # fungsi konversi magnitude -> elemen F (sudah diperbaiki)
    def gf_element_from_maybe_int(x):
        """
        Jika x sudah elemen lapangan F => kembalikan apa adanya.
        Jika x bisa dikonversi ke int (0..255) => kembalikan
int_to_GF256(int(x)).
        Jika tidak bisa keduanya, raise ValueError.
        """
        try:
            px = getattr(x, 'parent', None)
            if callable(px):
                try:
                    parent_of_x = x.parent()
                    if parent_of_x is F:
                        return x
                except Exception:
                    pass
            else:
                if px is F:
                    return x
        except Exception:
            pass

        try:
            xi = int(x)
        except Exception as e:
            raise ValueError(f"tidak bisa konversi magnitude {x!r} ke int atau
elemen F: {e}")
        if xi < 0 or xi > 255:
            raise ValueError("int magnitude harus di rentang 0..255")
        return int_to_GF256(xi)

    def add_errors_at_positions(cw, positions, magnitudes):
        """
        cw: list/sequence elemen F atau vector over F
        positions: list of int indices (0..n-1)
        magnitudes: list with same length; each magnitude int 0..255 atau elemen
F
        Mengembalikan: list elemen F (salinan cw dengan error additive di posisi
tsb).
        """
        if len(positions) != len(magnitudes):
            raise ValueError("positions dan magnitudes harus sama panjang")
        res = [c for c in list(cw)]
        n_local = len(res)
        for pos, mag in zip(positions, magnitudes):
            if not (0 <= pos < n_local):
                raise IndexError(f"pos {pos} out of range (0..{n_local-1})")
            mag_el = gf_element_from_maybe_int(mag)
            res[pos] = res[pos] + mag_el
        return res

    # fungsi bantu random errors (tetap bisa digunakan)
    def add_random_errors(cw, num_errors=1, allow_replacement=False,
mag_nonzero=True):
        n_local = len(cw)
        if not allow_replacement and num_errors > n_local:
            raise ValueError("num errors > n local dan allow replacement=False")
```

```
        if allow_replacement:
            positions = [ random.randrange(0, n_local) for _ in
range(num_errors) ]
        else:
            positions = random.sample(range(n_local), num_errors)
        magnitudes = []
        for _ in range(num_errors):
            m = 0
            while True:
                m = random.randint(1,255) if mag_nonzero else
random.randint(0,255)
                if not mag_nonzero or m != 0:
                    break
            magnitudes.append(m)
        res = add_errors_at_positions(cw, positions, magnitudes)
        return res, positions, magnitudes

    # gunakan posisi/magnitude dari daftar ep1/ev1 (sudah dipaksa FORCED_T
elemen)
    cw1_err = add_errors_to_cw(cw, ep1, ev1)
    cw2_err = add_errors_to_cw(cw2, ep2, ev2)

    print("\nCodeword1 asli (α^i):", [element_to_alpha_power(c) for c in cw])
    print("Codeword1 rusak (α^i):", [element_to_alpha_power(c) for c in
cw1_err])
    print("\nCodeword2 asli (α^i):", [element_to_alpha_power(c) for c in cw2])
    print("Codeword2 rusak (α^i):", [element_to_alpha_power(c) for c in
cw2_err])

    # diagnose & convert ambient jika diperlukan sebelum decode
    def diagnose_ambient_and_fix(word, C_obj, field_F, expected_n):
        diag = {}
        try:
            ambient = C_obj.ambient_space()
            diag['ambient_repr'] = ambient
        except Exception as e:
            diag['ambient_repr'] = f"Could not fetch ambient via
C_obj.ambient_space(): {e}"

        try:
            seq = list(word)
        except Exception:
            raise ValueError("word tidak bisa dikonversi menjadi list; berikan
list/vector.")

        diag['given_length'] = len(seq)
        diag['expected_length'] = expected_n

        converted = []
        problems = []
        for i, e in enumerate(seq):
            converted_el = None
            try:
                if hasattr(e, 'parent') and callable(e.parent):
                    try:
                        if e.parent() is field_F:
                            converted_el = e
                    except Exception:
                        pass
            except Exception:
                pass

            if converted_el is None:
                try:
                    ei = int(e)
                    if not (0 <= ei <= 255):
                        problems.append((i, e, "int out of range 0..255"))
                    else:
                        converted_el = int_to_GF256(ei)
                except Exception as ex:
                    problems.append((i, e, f"cannot cast to int: {ex}"))

            converted.append(converted_el)
```

```
        diag['conversion_problems'] = problems

        if len(converted) != expected_n:
            raise ValueError(f"Panjang word ({len(converted)}) != expected n
({expected_n}). "
                             "Pastikan word length = n. Diagnostics: " +
repr(diag))

        final_vec = vector(field_F, converted)
        diag['final_vector_parent'] = final_vec.parent()
        return final_vec, diag

    # contoh penggunaan diagnosa + decode via C.decode
    try:
        # use example 'corrupted' from earlier deterministic injection if
exists, else use cw1_err
        corrupted_for_decode = None
        try:
            corrupted_for_decode = corrupted  # from example injection earlier
(may or may not exist)
        except NameError:
            corrupted_for_decode = cw1_err

        fixed_vec, diagnostics = diagnose_ambient_and_fix(corrupted_for_decode,
C, F, n)
        print("Diagnostik:", diagnostics)
        corrected = C.decode(fixed_vec)
        print("\nDecode sukses. Hasil corrected codeword:")
        show_cw("corrected", corrected)
        print("corrected (hex):", [f"0x{poly_to_int(c.polynomial()):02x}" for c
in list(corrected)])
    except ValueError as e:
        print("ValueError saat persiapan decode:", e)
    except Exception as e:
        print("Error lain saat decode:", e)

    # 6) sindrom dan validasi
    t_capability = (n - k_ayat) // 2
    print(f"\n=== Hitung sindrom (2t = {2*t_capability}) dan validasi (evaluasi
& H•v^T) ===")
    S_eval_1 = syndromes_from_vector_sagemath(cw1_err, t_capability, step=5,
start=0)
    S_hv_1 = list(H * vector(F, cw1_err))
    print("\n-- Sindrom Codeword1 (evaluasi) --")
    for i, s in enumerate(S_eval_1, 1):
        print(f"S_{i} = {element_to_alpha_power(s)} (dec
{poly_to_int(s.polynomial())})")
    print("\n-- Sindrom Codeword1 (H•v^T) --")
    for i, s in enumerate(S_hv_1, 1):
        print(f"S_{i} = {element_to_alpha_power(s)} (dec
{poly_to_int(s.polynomial())})")
    same1 = all(S_eval_1[i] == S_hv_1[i] for i in range(len(S_eval_1)))
    print("Validasi sindrom codeword1:", "Cocok ✅" if same1 else "Tidak cocok
⚠")

    S_eval_2 = syndromes_from_vector_sagemath(cw2_err, t_capability, step=5,
start=0)
    S_hv_2 = list(H * vector(F, cw2_err))
    print("\n-- Sindrom Codeword2 (evaluasi) --")
    for i, s in enumerate(S_eval_2, 1):
        print(f"S_{i} = {element_to_alpha_power(s)} (dec
{poly_to_int(s.polynomial())})")
    print("\n-- Sindrom Codeword2 (H•v^T) --")
    for i, s in enumerate(S_hv_2, 1):
        print(f"S_{i} = {element_to_alpha_power(s)} (dec
{poly_to_int(s.polynomial())})")
    same2 = all(S_eval_2[i] == S_hv_2[i] for i in range(len(S_eval_2)))
    print("Validasi sindrom codeword2:", "Cocok ✅" if same2 else "Tidak cocok
⚠")

    # 7) decode: BM + Chien + Forney (menggunakan t_capability)
    print("\n=== Dekoding (Berlekamp-Massey -> Chien -> Forney) untuk codeword1
===")
```

```
    S1 = S_eval_1
    sigma1, L1 = berlekamp_massey(S1)
    print("Berlekamp-Massey hasil: degree L =", L1)
    print("sigma coeffs (α^i):", [element_to_alpha_power(c) for c in sigma1])
    errs_pos1 = chien_search(sigma1, n, step=5, start=0)
    print("Posisi error (Chien):", errs_pos1)
    mags1 = forney_magnitudes(S1, sigma1, errs_pos1, t_capability, step=5,
start=0)
    for pos, val in mags1.items():
        if val is None:
            print(f"pos {pos}: magnitude = ERROR(div0)")
        else:
            print(f"pos {pos}: magnitude = {element_to_alpha_power(val)} (dec
{poly_to_int(val.polynomial())})")
    # koreksi
    cw1_corr = list(cw1_err)
    for pos, mag in mags1.items():
        if mag is not None:
            cw1_corr[pos] -= mag
    print("\nSindrom setelah koreksi (codeword1):", [element_to_alpha_power(s)
for s in syndromes_from_vector_sagemath(cw1_corr, t_capability, step=5,
start=0)])
    print("Dekoding codeword1 sukses?" , "YA" if all(s==0 for s in
syndromes_from_vector_sagemath(cw1_corr, t_capability, step=5, start=0)) else
"TIDAK")

    print("\n=== Dekoding untuk codeword2 ===")
    S2 = S_eval_2
    sigma2, L2 = berlekamp_massey(S2)
    print("Berlekamp-Massey hasil: degree L =", L2)
    print("sigma coeffs (α^i):", [element_to_alpha_power(c) for c in sigma2])
    errs_pos2 = chien_search(sigma2, n, step=5, start=0)
    print("Posisi error (Chien):", errs_pos2)
    mags2 = forney_magnitudes(S2, sigma2, errs_pos2, t_capability, step=5,
start=0)
    for pos, val in mags2.items():
        if val is None:
            print(f"pos {pos}: magnitude = ERROR(div0)")
        else:
            print(f"pos {pos}: magnitude = {element_to_alpha_power(val)} (dec
{poly_to_int(val.polynomial())})")
    cw2_corr = list(cw2_err)
    for pos, mag in mags2.items():
        if mag is not None:
            cw2_corr[pos] -= mag
    print("\nSindrom setelah koreksi (codeword2):", [element_to_alpha_power(s)
for s in syndromes_from_vector_sagemath(cw2_corr, t_capability, step=5,
start=0)])
    print("Dekoding codeword2 sukses?" , "YA" if all(s==0 for s in
syndromes_from_vector_sagemath(cw2_corr, t_capability, step=5, start=0)) else
"TIDAK")

    # 8) tampil akhir
    show_cw("cw1 (rusak)", cw1_err)
    show_cw("cw1 (terkoreksi)", cw1_corr)
    show_cw("cw2 (rusak)", cw2_err)
    show_cw("cw2 (terkoreksi)", cw2_corr)

    try:
        fixed_vec, diagnostics = diagnose_ambient_and_fix(cw1_corr, C, F, n)
        print("Diagnostik:", diagnostics)
    except Exception as e:
        print("Diagnostik/gagal:", e)

    try:
        pw = C.decode_to_message(vector(F, [int_to_GF256(
poly_to_int(c.polynomial()) ) if hasattr(c,'polynomial') else
int_to_GF256(int(c)) for c in cw1_corr]))
        show_cw("pw1 (hasil decoding)", pw)
    except Exception as e:
        print("Gagal decode_to_message untuk cw1_corr:", e)

    try:
        fixed_vec2, diagnostics2 = diagnose_ambient_and_fix(cw2_corr, C, F, n)
```

```
        print("Diagnostik cw2:", diagnostics2)
    except Exception as e:
        print("Diagnostik cw2/gagal:", e)

    try:
        pw2 = C.decode_to_message(vector(F, [int_to_GF256(
poly_to_int(c.polynomial()) ) if hasattr(c,'polynomial') else
int_to_GF256(int(c)) for c in cw2_corr]))
        show_cw("pw2 (hasil decoding)", pw2)
    except Exception as e:
        print("Gagal decode_to_message untuk cw2_corr:", e)

# === SELESAI LOOP 10 AYAT ===
print("\n=== Semua ayat telah diproses (dengan 2 error per codeword, k = {0}).
===".format(k_ayat))
```

Lampiran 3. Matriks Generator dalam Bentuk Representasi α^i $GF(2^8)$

```
Generator matrix G (nrows=47, ncols=51):
r00 | ['α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0',
'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0',
'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0',
'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0',
'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0', 'α^0']
r01 | ['α^0', 'α^5', 'α^10', 'α^15', 'α^20', 'α^25', 'α^30', 'α^35', 'α^40',
'α^45', 'α^50', 'α^55', 'α^60', 'α^65', 'α^70', 'α^75', 'α^80', 'α^85', 'α^90',
'α^95', 'α^100', 'α^105', 'α^110', 'α^115', 'α^120', 'α^125', 'α^130', 'α^135',
'α^140', 'α^145', 'α^150', 'α^155', 'α^160', 'α^165', 'α^170', 'α^175', 'α^180',
'α^185', 'α^190', 'α^195', 'α^200', 'α^205', 'α^210', 'α^215', 'α^220', 'α^225',
'α^230', 'α^235', 'α^240', 'α^245', 'α^250']
r02 | ['α^0', 'α^10', 'α^20', 'α^30', 'α^40', 'α^50', 'α^60', 'α^70', 'α^80',
'α^90', 'α^100', 'α^110', 'α^120', 'α^130', 'α^140', 'α^150', 'α^160', 'α^170',
'α^180', 'α^190', 'α^200', 'α^210', 'α^220', 'α^230', 'α^240', 'α^250', 'α^5',
'α^15', 'α^25', 'α^35', 'α^45', 'α^55', 'α^65', 'α^75', 'α^85', 'α^95', 'α^105',
'α^115', 'α^125', 'α^135', 'α^145', 'α^155', 'α^165', 'α^175', 'α^185', 'α^195',
'α^205', 'α^215', 'α^225', 'α^235', 'α^245']
r03 | ['α^0', 'α^15', 'α^30', 'α^45', 'α^60', 'α^75', 'α^90', 'α^105', 'α^120',
'α^135', 'α^150', 'α^165', 'α^180', 'α^195', 'α^210', 'α^225', 'α^240', 'α^0',
'α^15', 'α^30', 'α^45', 'α^60', 'α^75', 'α^90', 'α^105', 'α^120', 'α^135',
'α^150', 'α^165', 'α^180', 'α^195', 'α^210', 'α^225', 'α^240', 'α^0', 'α^15',
'α^30', 'α^45', 'α^60', 'α^75', 'α^90', 'α^105', 'α^120', 'α^135', 'α^150',
'α^165', 'α^180', 'α^195', 'α^210', 'α^225', 'α^240']
r04 | ['α^0', 'α^20', 'α^40', 'α^60', 'α^80', 'α^100', 'α^120', 'α^140',
'α^160', 'α^180', 'α^200', 'α^220', 'α^240', 'α^5', 'α^25', 'α^45', 'α^65',
'α^85', 'α^105', 'α^125', 'α^145', 'α^165', 'α^185', 'α^205', 'α^225', 'α^245',
'α^10', 'α^30', 'α^50', 'α^70', 'α^90', 'α^110', 'α^130', 'α^150', 'α^170',
'α^190', 'α^210', 'α^230', 'α^250', 'α^15', 'α^35', 'α^55', 'α^75', 'α^95',
'α^115', 'α^135', 'α^155', 'α^175', 'α^195', 'α^215', 'α^235']
r05 | ['α^0', 'α^25', 'α^50', 'α^75', 'α^100', 'α^125', 'α^150', 'α^175',
'α^200', 'α^225', 'α^250', 'α^20', 'α^45', 'α^70', 'α^95', 'α^120', 'α^145',
'α^170', 'α^195', 'α^220', 'α^245', 'α^15', 'α^40', 'α^65', 'α^90', 'α^115',
'α^140', 'α^165', 'α^190', 'α^215', 'α^240', 'α^10', 'α^35', 'α^60', 'α^85',
'α^110', 'α^135', 'α^160', 'α^185', 'α^210', 'α^235', 'α^5', 'α^30', 'α^55',
'α^80', 'α^105', 'α^130', 'α^155', 'α^180', 'α^205', 'α^230']
r06 | ['α^0', 'α^30', 'α^60', 'α^90', 'α^120', 'α^150', 'α^180', 'α^210',
'α^240', 'α^15', 'α^45', 'α^75', 'α^105', 'α^135', 'α^165', 'α^195', 'α^225',
'α^0', 'α^30', 'α^60', 'α^90', 'α^120', 'α^150', 'α^180', 'α^210', 'α^240',
'α^15', 'α^45', 'α^75', 'α^105', 'α^135', 'α^165', 'α^195', 'α^225', 'α^0',
'α^30', 'α^60', 'α^90', 'α^120', 'α^150', 'α^180', 'α^210', 'α^240', 'α^15',
'α^45', 'α^75', 'α^105', 'α^135', 'α^165', 'α^195', 'α^225']
r07 | ['α^0', 'α^35', 'α^70', 'α^105', 'α^140', 'α^175', 'α^210', 'α^245',
'α^25', 'α^60', 'α^95', 'α^130', 'α^165', 'α^200', 'α^235', 'α^15', 'α^50',
'α^85', 'α^120', 'α^155', 'α^190', 'α^225', 'α^5', 'α^40', 'α^75', 'α^110',
'α^145', 'α^180', 'α^215', 'α^250', 'α^30', 'α^65', 'α^100', 'α^135', 'α^170',
'α^205', 'α^240', 'α^20', 'α^55', 'α^90', 'α^125', 'α^160', 'α^195', 'α^230',
'α^10', 'α^45', 'α^80', 'α^115', 'α^150', 'α^185', 'α^220']
r08 | ['α^0', 'α^40', 'α^80', 'α^120', 'α^160', 'α^200', 'α^240', 'α^25',
'α^65', 'α^105', 'α^145', 'α^185', 'α^225', 'α^10', 'α^50', 'α^90', 'α^130',
'α^170', 'α^210', 'α^250', 'α^35', 'α^75', 'α^115', 'α^155', 'α^195', 'α^235',
'α^20', 'α^60', 'α^100', 'α^140', 'α^180', 'α^220', 'α^5', 'α^45', 'α^85',
'α^125', 'α^165', 'α^205', 'α^245', 'α^30', 'α^70', 'α^110', 'α^150', 'α^190',
'α^230', 'α^15', 'α^55', 'α^95', 'α^135', 'α^175', 'α^215']
r09 | ['α^0', 'α^45', 'α^90', 'α^135', 'α^180', 'α^225', 'α^15', 'α^60',
'α^105', 'α^150', 'α^195', 'α^240', 'α^30', 'α^75', 'α^120', 'α^165', 'α^210',
'α^0', 'α^45', 'α^90', 'α^135', 'α^180', 'α^225', 'α^15', 'α^60', 'α^105',
'α^150', 'α^195', 'α^240', 'α^30', 'α^75', 'α^120', 'α^165', 'α^210', 'α^0',
'α^45', 'α^90', 'α^135', 'α^180', 'α^225', 'α^15', 'α^60', 'α^105', 'α^150',
'α^195', 'α^240', 'α^30', 'α^75', 'α^120', 'α^165', 'α^210']
r10 | ['α^0', 'α^50', 'α^100', 'α^150', 'α^200', 'α^250', 'α^45', 'α^95',
'α^145', 'α^195', 'α^245', 'α^40', 'α^90', 'α^140', 'α^190', 'α^240', 'α^35',
'α^85', 'α^135', 'α^185', 'α^235', 'α^30', 'α^80', 'α^130', 'α^180', 'α^230',
'α^25', 'α^75', 'α^125', 'α^175', 'α^225', 'α^20', 'α^70', 'α^120', 'α^170',
'α^220', 'α^15', 'α^65', 'α^115', 'α^165', 'α^215', 'α^10', 'α^60', 'α^110',
'α^160', 'α^210', 'α^5', 'α^55', 'α^105', 'α^155', 'α^205']
r11 | ['α^0', 'α^55', 'α^110', 'α^165', 'α^220', 'α^20', 'α^75', 'α^130',
'α^185', 'α^240', 'α^40', 'α^95', 'α^150', 'α^205', 'α^5', 'α^60', 'α^115',
'α^170', 'α^225', 'α^25', 'α^80', 'α^135', 'α^190', 'α^245', 'α^45', 'α^100',
'α^155', 'α^210', 'α^10', 'α^65', 'α^120', 'α^175', 'α^230', 'α^30', 'α^85',
'α^140', 'α^195', 'α^250', 'α^50', 'α^105', 'α^160', 'α^215', 'α^15', 'α^70',
'α^125', 'α^180', 'α^235', 'α^35', 'α^90', 'α^145', 'α^200']
```

r12 | ['α^0', 'α^60', 'α^120', 'α^180', 'α^240', 'α^45', 'α^105', 'α^165',
'α^225', 'α^30', 'α^90', 'α^150', 'α^210', 'α^15', 'α^75', 'α^135', 'α^195',
'α^0', 'α^60', 'α^120', 'α^180', 'α^240', 'α^45', 'α^105', 'α^165', 'α^225',
'α^30', 'α^90', 'α^150', 'α^210', 'α^15', 'α^75', 'α^135', 'α^195', 'α^0',
'α^60', 'α^120', 'α^180', 'α^240', 'α^45', 'α^105', 'α^165', 'α^225', 'α^30',
'α^90', 'α^150', 'α^210', 'α^15', 'α^75', 'α^135', 'α^195']
r13 | ['α^0', 'α^65', 'α^130', 'α^195', 'α^5', 'α^70', 'α^135', 'α^200', 'α^10',
'α^75', 'α^140', 'α^205', 'α^15', 'α^80', 'α^145', 'α^210', 'α^20', 'α^85',
'α^150', 'α^215', 'α^25', 'α^90', 'α^155', 'α^220', 'α^30', 'α^95', 'α^160',
'α^225', 'α^35', 'α^100', 'α^165', 'α^230', 'α^40', 'α^105', 'α^170', 'α^235',
'α^45', 'α^110', 'α^175', 'α^240', 'α^50', 'α^115', 'α^180', 'α^245', 'α^55',
'α^120', 'α^185', 'α^250', 'α^60', 'α^125', 'α^190']
r14 | ['α^0', 'α^70', 'α^140', 'α^210', 'α^25', 'α^95', 'α^165', 'α^235',
'α^50', 'α^120', 'α^190', 'α^5', 'α^75', 'α^145', 'α^215', 'α^30', 'α^100',
'α^170', 'α^240', 'α^55', 'α^125', 'α^195', 'α^10', 'α^80', 'α^150', 'α^220',
'α^35', 'α^105', 'α^175', 'α^245', 'α^60', 'α^130', 'α^200', 'α^15', 'α^85',
'α^155', 'α^225', 'α^40', 'α^110', 'α^180', 'α^250', 'α^65', 'α^135', 'α^205',
'α^20', 'α^90', 'α^160', 'α^230', 'α^45', 'α^115', 'α^185']
r15 | ['α^0', 'α^75', 'α^150', 'α^225', 'α^45', 'α^120', 'α^195', 'α^15',
'α^90', 'α^165', 'α^240', 'α^60', 'α^135', 'α^210', 'α^30', 'α^105', 'α^180',
'α^0', 'α^75', 'α^150', 'α^225', 'α^45', 'α^120', 'α^195', 'α^15', 'α^90',
'α^165', 'α^240', 'α^60', 'α^135', 'α^210', 'α^30', 'α^105', 'α^180', 'α^0',
'α^75', 'α^150', 'α^225', 'α^45', 'α^120', 'α^195', 'α^15', 'α^90', 'α^165',
'α^240', 'α^60', 'α^135', 'α^210', 'α^30', 'α^105', 'α^180']
r16 | ['α^0', 'α^80', 'α^160', 'α^240', 'α^65', 'α^145', 'α^225', 'α^50',
'α^130', 'α^210', 'α^35', 'α^115', 'α^195', 'α^20', 'α^100', 'α^180', 'α^5',
'α^85', 'α^165', 'α^245', 'α^70', 'α^150', 'α^230', 'α^55', 'α^135', 'α^215',
'α^40', 'α^120', 'α^200', 'α^25', 'α^105', 'α^185', 'α^10', 'α^90', 'α^170',
'α^250', 'α^75', 'α^155', 'α^235', 'α^60', 'α^140', 'α^220', 'α^45', 'α^125',
'α^205', 'α^30', 'α^110', 'α^190', 'α^15', 'α^95', 'α^175']
r17 | ['α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170',
'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0',
'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85',
'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170',
'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0', 'α^85', 'α^170', 'α^0',
'α^85', 'α^170']
r18 | ['α^0', 'α^90', 'α^180', 'α^15', 'α^105', 'α^195', 'α^30', 'α^120',
'α^210', 'α^45', 'α^135', 'α^225', 'α^60', 'α^150', 'α^240', 'α^75', 'α^165',
'α^0', 'α^90', 'α^180', 'α^15', 'α^105', 'α^195', 'α^30', 'α^120', 'α^210',
'α^45', 'α^135', 'α^225', 'α^60', 'α^150', 'α^240', 'α^75', 'α^165', 'α^0',
'α^90', 'α^180', 'α^15', 'α^105', 'α^195', 'α^30', 'α^120', 'α^210', 'α^45',
'α^135', 'α^225', 'α^60', 'α^150', 'α^240', 'α^75', 'α^165']
r19 | ['α^0', 'α^95', 'α^190', 'α^30', 'α^125', 'α^220', 'α^60', 'α^155',
'α^250', 'α^90', 'α^185', 'α^25', 'α^120', 'α^215', 'α^55', 'α^150', 'α^245',
'α^85', 'α^180', 'α^20', 'α^115', 'α^210', 'α^50', 'α^145', 'α^240', 'α^80',
'α^175', 'α^15', 'α^110', 'α^205', 'α^45', 'α^140', 'α^235', 'α^75', 'α^170',
'α^10', 'α^105', 'α^200', 'α^40', 'α^135', 'α^230', 'α^70', 'α^165', 'α^5',
'α^100', 'α^195', 'α^35', 'α^130', 'α^225', 'α^65', 'α^160']
r20 | ['α^0', 'α^100', 'α^200', 'α^45', 'α^145', 'α^245', 'α^90', 'α^190',
'α^35', 'α^135', 'α^235', 'α^80', 'α^180', 'α^25', 'α^125', 'α^225', 'α^70',
'α^170', 'α^15', 'α^115', 'α^215', 'α^60', 'α^160', 'α^5', 'α^105', 'α^205',
'α^50', 'α^150', 'α^250', 'α^95', 'α^195', 'α^40', 'α^140', 'α^240', 'α^85',
'α^185', 'α^30', 'α^130', 'α^230', 'α^75', 'α^175', 'α^20', 'α^120', 'α^220',
'α^65', 'α^165', 'α^10', 'α^110', 'α^210', 'α^55', 'α^155']
r21 | ['α^0', 'α^105', 'α^210', 'α^60', 'α^165', 'α^15', 'α^120', 'α^225',
'α^75', 'α^180', 'α^30', 'α^135', 'α^240', 'α^90', 'α^195', 'α^45', 'α^150',
'α^0', 'α^105', 'α^210', 'α^60', 'α^165', 'α^15', 'α^120', 'α^225', 'α^75',
'α^180', 'α^30', 'α^135', 'α^240', 'α^90', 'α^195', 'α^45', 'α^150', 'α^0',
'α^105', 'α^210', 'α^60', 'α^165', 'α^15', 'α^120', 'α^225', 'α^75', 'α^180',
'α^30', 'α^135', 'α^240', 'α^90', 'α^195', 'α^45', 'α^150']
r22 | ['α^0', 'α^110', 'α^220', 'α^75', 'α^185', 'α^40', 'α^150', 'α^5',
'α^115', 'α^225', 'α^80', 'α^190', 'α^45', 'α^155', 'α^10', 'α^120', 'α^230',
'α^85', 'α^195', 'α^50', 'α^160', 'α^15', 'α^125', 'α^235', 'α^90', 'α^200',
'α^55', 'α^165', 'α^20', 'α^130', 'α^240', 'α^95', 'α^205', 'α^60', 'α^170',
'α^25', 'α^135', 'α^245', 'α^100', 'α^210', 'α^65', 'α^175', 'α^30', 'α^140',
'α^250', 'α^105', 'α^215', 'α^70', 'α^180', 'α^35', 'α^145']
r23 | ['α^0', 'α^115', 'α^230', 'α^90', 'α^205', 'α^65', 'α^180', 'α^40',
'α^155', 'α^15', 'α^130', 'α^245', 'α^105', 'α^220', 'α^80', 'α^195', 'α^55',
'α^170', 'α^30', 'α^145', 'α^5', 'α^120', 'α^235', 'α^95', 'α^210', 'α^70',
'α^185', 'α^45', 'α^160', 'α^20', 'α^135', 'α^250', 'α^110', 'α^225', 'α^85',
'α^200', 'α^60', 'α^175', 'α^35', 'α^150', 'α^10', 'α^125', 'α^240', 'α^100',
'α^215', 'α^75', 'α^190', 'α^50', 'α^165', 'α^25', 'α^140']
r24 | ['α^0', 'α^120', 'α^240', 'α^105', 'α^225', 'α^90', 'α^210', 'α^75',
'α^195', 'α^60', 'α^180', 'α^45', 'α^165', 'α^30', 'α^150', 'α^15', 'α^135',

'α^0', 'α^120', 'α^240', 'α^105', 'α^225', 'α^90', 'α^210', 'α^75', 'α^195',
'α^60', 'α^180', 'α^45', 'α^165', 'α^30', 'α^150', 'α^15', 'α^135', 'α^0',
'α^120', 'α^240', 'α^105', 'α^225', 'α^90', 'α^210', 'α^75', 'α^195', 'α^60',
'α^180', 'α^45', 'α^165', 'α^30', 'α^150', 'α^15', 'α^135']

r25 | ['α^0', 'α^125', 'α^250', 'α^120', 'α^245', 'α^115', 'α^240', 'α^110',
'α^235', 'α^105', 'α^230', 'α^100', 'α^225', 'α^95', 'α^220', 'α^90', 'α^215',
'α^85', 'α^210', 'α^80', 'α^205', 'α^75', 'α^200', 'α^70', 'α^195', 'α^65',
'α^190', 'α^60', 'α^185', 'α^55', 'α^180', 'α^50', 'α^175', 'α^45', 'α^170',
'α^40', 'α^165', 'α^35', 'α^160', 'α^30', 'α^155', 'α^25', 'α^150', 'α^20',
'α^145', 'α^15', 'α^140', 'α^10', 'α^135', 'α^5', 'α^130']

r26 | ['α^0', 'α^130', 'α^5', 'α^135', 'α^10', 'α^140', 'α^15', 'α^145', 'α^20',
'α^150', 'α^25', 'α^155', 'α^30', 'α^160', 'α^35', 'α^165', 'α^40', 'α^170',
'α^45', 'α^175', 'α^50', 'α^180', 'α^55', 'α^185', 'α^60', 'α^190', 'α^65',
'α^195', 'α^70', 'α^200', 'α^75', 'α^205', 'α^80', 'α^210', 'α^85', 'α^215',
'α^90', 'α^220', 'α^95', 'α^225', 'α^100', 'α^230', 'α^105', 'α^235', 'α^110',
'α^240', 'α^115', 'α^245', 'α^120', 'α^250', 'α^125']

r27 | ['α^0', 'α^135', 'α^15', 'α^150', 'α^30', 'α^165', 'α^45', 'α^180',
'α^60', 'α^195', 'α^75', 'α^210', 'α^90', 'α^225', 'α^105', 'α^240', 'α^120',
'α^0', 'α^135', 'α^15', 'α^150', 'α^30', 'α^165', 'α^45', 'α^180', 'α^60',
'α^195', 'α^75', 'α^210', 'α^90', 'α^225', 'α^105', 'α^240', 'α^120', 'α^0',
'α^135', 'α^15', 'α^150', 'α^30', 'α^165', 'α^45', 'α^180', 'α^60', 'α^195',
'α^75', 'α^210', 'α^90', 'α^225', 'α^105', 'α^240', 'α^120']

r28 | ['α^0', 'α^140', 'α^25', 'α^165', 'α^50', 'α^190', 'α^75', 'α^215',
'α^100', 'α^240', 'α^125', 'α^10', 'α^150', 'α^35', 'α^175', 'α^60', 'α^200',
'α^85', 'α^225', 'α^110', 'α^250', 'α^135', 'α^20', 'α^160', 'α^45', 'α^185',
'α^70', 'α^210', 'α^95', 'α^235', 'α^120', 'α^5', 'α^145', 'α^30', 'α^170',
'α^55', 'α^195', 'α^80', 'α^220', 'α^105', 'α^245', 'α^130', 'α^15', 'α^155',
'α^40', 'α^180', 'α^65', 'α^205', 'α^90', 'α^230', 'α^115']

r29 | ['α^0', 'α^145', 'α^35', 'α^180', 'α^70', 'α^215', 'α^105', 'α^250',
'α^140', 'α^30', 'α^175', 'α^65', 'α^210', 'α^100', 'α^245', 'α^135', 'α^25',
'α^170', 'α^60', 'α^205', 'α^95', 'α^240', 'α^130', 'α^20', 'α^165', 'α^55',
'α^200', 'α^90', 'α^235', 'α^125', 'α^15', 'α^160', 'α^50', 'α^195', 'α^85',
'α^230', 'α^120', 'α^10', 'α^155', 'α^45', 'α^190', 'α^80', 'α^225', 'α^115',
'α^5', 'α^150', 'α^40', 'α^185', 'α^75', 'α^220', 'α^110']

r30 | ['α^0', 'α^150', 'α^45', 'α^195', 'α^90', 'α^240', 'α^135', 'α^30',
'α^180', 'α^75', 'α^225', 'α^120', 'α^15', 'α^165', 'α^60', 'α^210', 'α^105',
'α^0', 'α^150', 'α^45', 'α^195', 'α^90', 'α^240', 'α^135', 'α^30', 'α^180',
'α^75', 'α^225', 'α^120', 'α^15', 'α^165', 'α^60', 'α^210', 'α^105', 'α^0',
'α^150', 'α^45', 'α^195', 'α^90', 'α^240', 'α^135', 'α^30', 'α^180', 'α^75',
'α^225', 'α^120', 'α^15', 'α^165', 'α^60', 'α^210', 'α^105']

r31 | ['α^0', 'α^155', 'α^55', 'α^210', 'α^110', 'α^10', 'α^165', 'α^65',
'α^220', 'α^120', 'α^20', 'α^175', 'α^75', 'α^230', 'α^130', 'α^30', 'α^185',
'α^85', 'α^240', 'α^140', 'α^40', 'α^195', 'α^95', 'α^250', 'α^150', 'α^50',
'α^205', 'α^105', 'α^5', 'α^160', 'α^60', 'α^215', 'α^115', 'α^15', 'α^170',
'α^70', 'α^225', 'α^125', 'α^25', 'α^180', 'α^80', 'α^235', 'α^135', 'α^35',
'α^190', 'α^90', 'α^245', 'α^145', 'α^45', 'α^200', 'α^100']

r32 | ['α^0', 'α^160', 'α^65', 'α^225', 'α^130', 'α^35', 'α^195', 'α^100',
'α^5', 'α^165', 'α^70', 'α^230', 'α^135', 'α^40', 'α^200', 'α^105', 'α^10',
'α^170', 'α^75', 'α^235', 'α^140', 'α^45', 'α^205', 'α^110', 'α^15', 'α^175',
'α^80', 'α^240', 'α^145', 'α^50', 'α^210', 'α^115', 'α^20', 'α^180', 'α^85',
'α^245', 'α^150', 'α^55', 'α^215', 'α^120', 'α^25', 'α^185', 'α^90', 'α^250',
'α^155', 'α^60', 'α^220', 'α^125', 'α^30', 'α^190', 'α^95']

r33 | ['α^0', 'α^165', 'α^75', 'α^240', 'α^150', 'α^60', 'α^225', 'α^135',
'α^45', 'α^210', 'α^120', 'α^30', 'α^195', 'α^105', 'α^15', 'α^180', 'α^90',
'α^0', 'α^165', 'α^75', 'α^240', 'α^150', 'α^60', 'α^225', 'α^135', 'α^45',
'α^210', 'α^120', 'α^30', 'α^195', 'α^105', 'α^15', 'α^180', 'α^90', 'α^0',
'α^165', 'α^75', 'α^240', 'α^150', 'α^60', 'α^225', 'α^135', 'α^45', 'α^210',
'α^120', 'α^30', 'α^195', 'α^105', 'α^15', 'α^180', 'α^90']

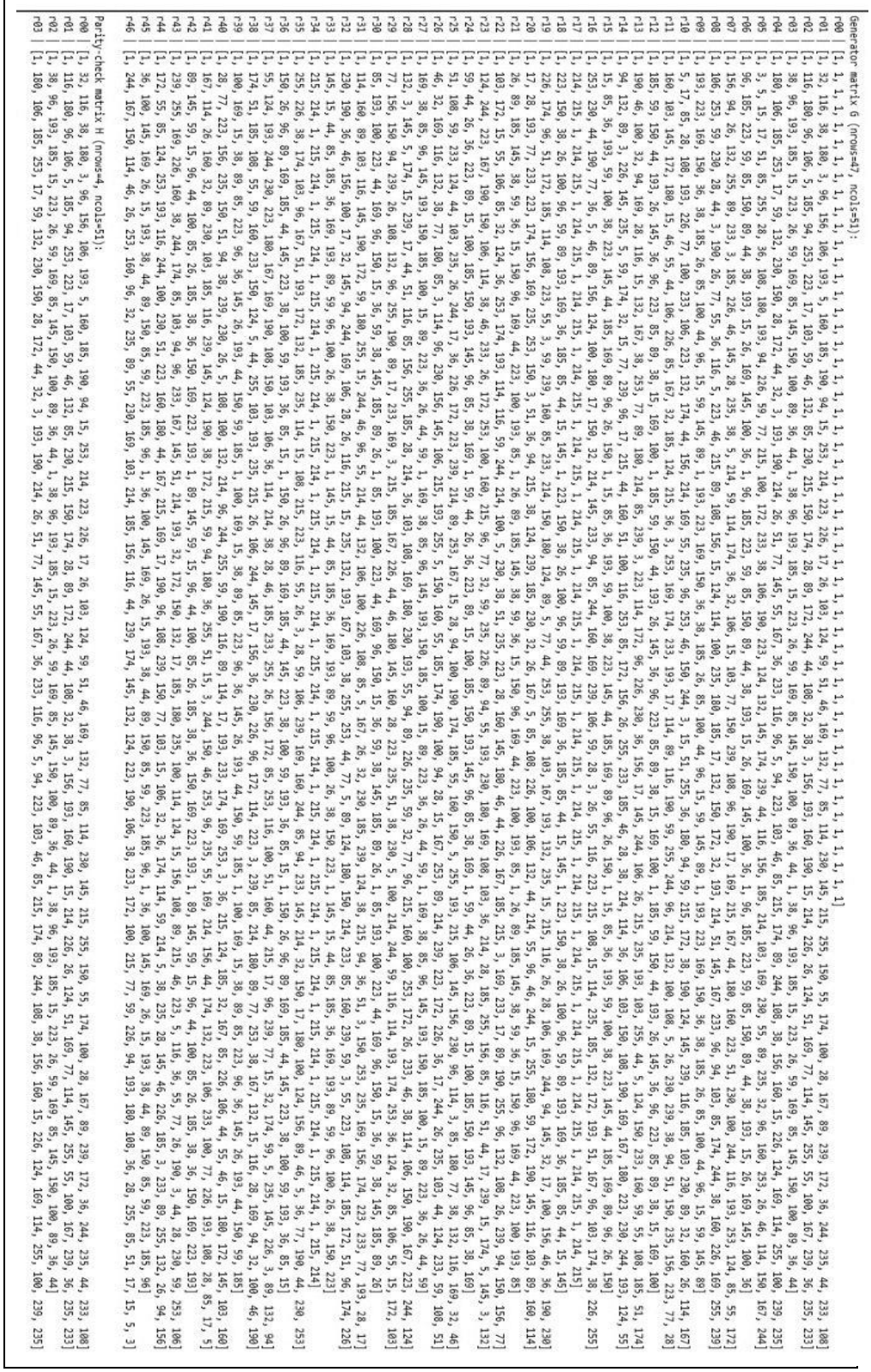
r34 | ['α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85',
'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0',
'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170',
'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85',
'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0', 'α^170', 'α^85', 'α^0',
'α^170', 'α^85']

r35 | ['α^0', 'α^175', 'α^95', 'α^15', 'α^190', 'α^110', 'α^30', 'α^205',
'α^125', 'α^45', 'α^220', 'α^140', 'α^60', 'α^235', 'α^155', 'α^75', 'α^250',
'α^170', 'α^90', 'α^10', 'α^185', 'α^105', 'α^25', 'α^200', 'α^120', 'α^40',
'α^215', 'α^135', 'α^55', 'α^230', 'α^150', 'α^70', 'α^245', 'α^165', 'α^85',
'α^5', 'α^180', 'α^100', 'α^20', 'α^195', 'α^115', 'α^35', 'α^210', 'α^130',
'α^50', 'α^225', 'α^145', 'α^65', 'α^240', 'α^160', 'α^80']

r36 | ['α^0', 'α^180', 'α^105', 'α^30', 'α^210', 'α^135', 'α^60', 'α^240',
'α^165', 'α^90', 'α^15', 'α^195', 'α^120', 'α^45', 'α^225', 'α^150', 'α^75',
'α^0', 'α^180', 'α^105', 'α^30', 'α^210', 'α^135', 'α^60', 'α^240', 'α^165',
'α^90', 'α^15', 'α^195', 'α^120', 'α^45', 'α^225', 'α^150', 'α^75', 'α^0',

'α^180', 'α^105', 'α^30', 'α^210', 'α^135', 'α^60', 'α^240', 'α^165', 'α^90',
'α^15', 'α^195', 'α^120', 'α^45', 'α^225', 'α^150', 'α^75']
r37 | ['α^0', 'α^185', 'α^115', 'α^45', 'α^230', 'α^160', 'α^90', 'α^20',
'α^205', 'α^135', 'α^65', 'α^250', 'α^180', 'α^110', 'α^40', 'α^225', 'α^155',
'α^85', 'α^15', 'α^200', 'α^130', 'α^60', 'α^245', 'α^175', 'α^105', 'α^35',
'α^220', 'α^150', 'α^80', 'α^10', 'α^195', 'α^125', 'α^55', 'α^240', 'α^170',
'α^100', 'α^30', 'α^215', 'α^145', 'α^75', 'α^5', 'α^190', 'α^120', 'α^50',
'α^235', 'α^165', 'α^95', 'α^25', 'α^210', 'α^140', 'α^70']
r38 | ['α^0', 'α^190', 'α^125', 'α^60', 'α^250', 'α^185', 'α^120', 'α^55',
'α^245', 'α^180', 'α^115', 'α^50', 'α^240', 'α^175', 'α^110', 'α^45', 'α^235',
'α^170', 'α^105', 'α^40', 'α^230', 'α^165', 'α^100', 'α^35', 'α^225', 'α^160',
'α^95', 'α^30', 'α^220', 'α^155', 'α^90', 'α^25', 'α^215', 'α^150', 'α^85',
'α^20', 'α^210', 'α^145', 'α^80', 'α^15', 'α^205', 'α^140', 'α^75', 'α^10',
'α^200', 'α^135', 'α^70', 'α^5', 'α^195', 'α^130', 'α^65']
r39 | ['α^0', 'α^195', 'α^135', 'α^75', 'α^15', 'α^210', 'α^150', 'α^90',
'α^30', 'α^225', 'α^165', 'α^105', 'α^45', 'α^240', 'α^180', 'α^120', 'α^60',
'α^0', 'α^195', 'α^135', 'α^75', 'α^15', 'α^210', 'α^150', 'α^90', 'α^30',
'α^225', 'α^165', 'α^105', 'α^45', 'α^240', 'α^180', 'α^120', 'α^60', 'α^0',
'α^195', 'α^135', 'α^75', 'α^15', 'α^210', 'α^150', 'α^90', 'α^30', 'α^225',
'α^165', 'α^105', 'α^45', 'α^240', 'α^180', 'α^120', 'α^60']
r40 | ['α^0', 'α^200', 'α^145', 'α^90', 'α^35', 'α^235', 'α^180', 'α^125',
'α^70', 'α^15', 'α^215', 'α^160', 'α^105', 'α^50', 'α^250', 'α^195', 'α^140',
'α^85', 'α^30', 'α^230', 'α^175', 'α^120', 'α^65', 'α^10', 'α^210', 'α^155',
'α^100', 'α^45', 'α^245', 'α^190', 'α^135', 'α^80', 'α^25', 'α^225', 'α^170',
'α^115', 'α^60', 'α^5', 'α^205', 'α^150', 'α^95', 'α^40', 'α^240', 'α^185',
'α^130', 'α^75', 'α^20', 'α^220', 'α^165', 'α^110', 'α^55']
r41 | ['α^0', 'α^205', 'α^155', 'α^105', 'α^55', 'α^5', 'α^210', 'α^160',
'α^110', 'α^60', 'α^10', 'α^215', 'α^165', 'α^115', 'α^65', 'α^15', 'α^220',
'α^170', 'α^120', 'α^70', 'α^20', 'α^225', 'α^175', 'α^125', 'α^75', 'α^25',
'α^230', 'α^180', 'α^130', 'α^80', 'α^30', 'α^235', 'α^185', 'α^135', 'α^85',
'α^35', 'α^240', 'α^190', 'α^140', 'α^90', 'α^40', 'α^245', 'α^195', 'α^145',
'α^95', 'α^45', 'α^250', 'α^200', 'α^150', 'α^100', 'α^50']
r42 | ['α^0', 'α^210', 'α^165', 'α^120', 'α^75', 'α^30', 'α^240', 'α^195',
'α^150', 'α^105', 'α^60', 'α^15', 'α^225', 'α^180', 'α^135', 'α^90', 'α^45',
'α^0', 'α^210', 'α^165', 'α^120', 'α^75', 'α^30', 'α^240', 'α^195', 'α^150',
'α^105', 'α^60', 'α^15', 'α^225', 'α^180', 'α^135', 'α^90', 'α^45', 'α^0',
'α^210', 'α^165', 'α^120', 'α^75', 'α^30', 'α^240', 'α^195', 'α^150', 'α^105',
'α^60', 'α^15', 'α^225', 'α^180', 'α^135', 'α^90', 'α^45']
r43 | ['α^0', 'α^215', 'α^175', 'α^135', 'α^95', 'α^55', 'α^15', 'α^230',
'α^190', 'α^150', 'α^110', 'α^70', 'α^30', 'α^245', 'α^205', 'α^165', 'α^125',
'α^85', 'α^45', 'α^5', 'α^220', 'α^180', 'α^140', 'α^100', 'α^60', 'α^20',
'α^235', 'α^195', 'α^155', 'α^115', 'α^75', 'α^35', 'α^250', 'α^210', 'α^170',
'α^130', 'α^90', 'α^50', 'α^10', 'α^225', 'α^185', 'α^145', 'α^105', 'α^65',
'α^25', 'α^240', 'α^200', 'α^160', 'α^120', 'α^80', 'α^40']
r44 | ['α^0', 'α^220', 'α^185', 'α^150', 'α^115', 'α^80', 'α^45', 'α^10',
'α^230', 'α^195', 'α^160', 'α^125', 'α^90', 'α^55', 'α^20', 'α^240', 'α^205',
'α^170', 'α^135', 'α^100', 'α^65', 'α^30', 'α^250', 'α^215', 'α^180', 'α^145',
'α^110', 'α^75', 'α^40', 'α^5', 'α^225', 'α^190', 'α^155', 'α^120', 'α^85',
'α^50', 'α^15', 'α^235', 'α^200', 'α^165', 'α^130', 'α^95', 'α^60', 'α^25',
'α^245', 'α^210', 'α^175', 'α^140', 'α^105', 'α^70', 'α^35']
r45 | ['α^0', 'α^225', 'α^195', 'α^165', 'α^135', 'α^105', 'α^75', 'α^45',
'α^15', 'α^240', 'α^210', 'α^180', 'α^150', 'α^120', 'α^90', 'α^60', 'α^30',
'α^0', 'α^225', 'α^195', 'α^165', 'α^135', 'α^105', 'α^75', 'α^45', 'α^15',
'α^240', 'α^210', 'α^180', 'α^150', 'α^120', 'α^90', 'α^60', 'α^30', 'α^0',
'α^225', 'α^195', 'α^165', 'α^135', 'α^105', 'α^75', 'α^45', 'α^15', 'α^240',
'α^210', 'α^180', 'α^150', 'α^120', 'α^90', 'α^60', 'α^30']
r46 | ['α^0', 'α^230', 'α^205', 'α^180', 'α^155', 'α^130', 'α^105', 'α^80',
'α^55', 'α^30', 'α^5', 'α^235', 'α^210', 'α^185', 'α^160', 'α^135', 'α^110',
'α^85', 'α^60', 'α^35', 'α^10', 'α^240', 'α^215', 'α^190', 'α^165', 'α^140',
'α^115', 'α^90', 'α^65', 'α^40', 'α^15', 'α^245', 'α^220', 'α^195', 'α^170',
'α^145', 'α^120', 'α^95', 'α^70', 'α^45', 'α^20', 'α^250', 'α^225', 'α^200',
'α^175', 'α^150', 'α^125', 'α^100', 'α^75', 'α^50', 'α^25']

Lampiran 4. Matriks Generator dan *Parity-Check* dalam Bentuk Desimal



RIWAYAT HIDUP

Tahira Khuwalidia Shabirah Ma'ruf lahir di Pasuruan pada tanggal 30 Maret 2003. Penulis berasal dari keluarga yang senantiasa menanamkan nilai-nilai pendidikan, kedisiplinan, dan keimanan sebagai bekal utama dalam kehidupan. Pendidikan formal penulis dimulai di TK Sri Wiji Handayani, kemudian dilanjutkan di SDN Pucangsari 1 hingga lulus pada tahun 2015. Selanjutnya, penulis menempuh pendidikan di SMP Modern Al-Rifai'e 2 dan lulus pada tahun 2018, kemudian melanjutkan pendidikan menengah atas di SMA Ma'arif NU Pandaan dan lulus pada tahun 2021. Pada tahun yang sama, penulis melanjutkan studi di Universitas Islam Negeri Maulana Malik Ibrahim Malang, Program Studi Matematika, Fakultas Sains dan Teknologi. Selama masa perkuliahan, penulis aktif dalam berbagai kegiatan akademik dan organisasi kemahasiswaan, antara lain sebagai asisten praktikum mata kuliah Analisis Numerik, tergabung dalam Himpunan Mahasiswa Program Studi (HMPS) Integral Matematika, serta menjabat sebagai Ketua Divisi Fundraising pada kegiatan KOMET XXII. Selain itu, penulis juga mengikuti berbagai kegiatan organisasi lain yang mendukung pengembangan diri. Di luar kegiatan akademik, penulis mengikuti program magang di PT Icon Plus PLN Surabaya serta aktif mengajar bimbingan belajar di Elfaza. Berbagai pengalaman tersebut menjadi bekal penting bagi penulis dalam membentuk karakter, memperluas wawasan, serta mempersiapkan diri untuk berkontribusi secara profesional maupun sosial di masa mendatang.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

BUKTI KONSULTASI SKRIPSI

Nama : Tahira Khuwalidia Shabirah Ma'ruf
NIM : 210601110087
Fakultas / Jurusan : Sains dan Teknologi / Matematika
Judul Skripsi : Implementasi Kode Reed-Solomon untuk Deteksi dan Koreksi Kesalahan Transmisi Ayat Al-Qur'an Menggunakan Pengkodean Huruf Hijaiyah
Pembimbing I : Muhammad Khudzaifah, M.Si.
Pembimbing II : Dr. Fachrur Rozi, M.Si.

No	Tanggal	Hal	Tanda Tangan
1.	7 November 2024	Konsultasi Bab I, II, dan III	1.
2.	29 April 2025	Konsultasi Bab I, II, dan III	2.
3.	14 Mei 2025	Konsultasi Bab I, II, dan III	3.
4.	5 Juni 2025	ACC Bab I, II, dan III	4.
5.	10 Juni 2025	Konsultasi Kajian Agama Bab I dan II	5.
6.	16 Juni 2025	Konsultasi Kajian Agama Bab I dan II	6.
7.	18 Juni 2025	Konsultasi Kajian Agama Bab I dan II	7.
8.	18 Juni 2025	ACC Kajian Agama Bab I dan II	8.
9.	29 Agustus 2025	ACC Seminar Proposal	9.
10.	12 November 2025	Konsultasi Revisi Seminar Proposal	10.
11.	20 November 2025	Konsultasi Bab IV	11.
12.	25 November 2025	Konsultasi Bab IV	12.
13.	26 November 2025	Konsultasi Kajian Agama Bab IV	13.
14.	27 November 2025	Konsultasi Kajian Agama Bab IV	14.
15.	27 November 2025	ACC Kajian Agama Bab IV	15.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

16.	28 November 2025	Konsultasi Bab IV dan V	16.
17.	28 November 2025	ACC Bab IV dan V	17.
18.	28 November 2025	ACC Seminar Hasil	18.
19.	9 Desember 2025	Konsultasi Revisi Seminar Hasil	19.
20.	15 Desember 2025	Sidang Skripsi	20.
21	22 Desember 2025	ACC Keseluruhan	21.

Malang, 22 Desember 2025
Mengetahui,
Ketua Program Studi Matematika

Dr. Fachrur Rozi, M.Si.
NIP. 19800527 200801 1 012