PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK PROTEKSI DATA CITRA DENGAN MENGGUNAKAN METODE WATERMARKING

TUGAS AKHIR

Disusun Oleh :

ACHMAD HUSIN
NRP. 2693100036

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2001

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK PROTEKSI DATA CITRA DENGAN MENGGUNAKAN METODE WATERMARKING

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Esther Hanaya, M.Sc.	**Rully Soelaiman, S.Kom.**
NIP. 130 816 212	**NIP. 132 085 802**

SURABAYA
Agustus, 2001

"Bacalah dengan menyebut nama Tuhanmu, Yang menciptakan. Dia telah menciptakan manusia dari segumpal darah. Bacalah! Dan Tuhanmu lah yang paling Pemurah. Yang telah mengajar (manusia) dengan perantaraan kalam. Dia telah mengajarkan kepada manusia apa yang tidak diketahuinya."

(QS. 96:1—5)

..... Katakanlah : "Adakah sama orang-orang yang mengetahui dengan orang-orang yang tidak mengetahui? Sesungguhnya orang yang berakallah yang dapat menerima pelajaran."

(QS. 39:9)

"..... Allah meninggikan orang yang beriman di antara kamu dan orang-orang yang diberi ilmu pengetahuan, beberapa derajat ....."

(QS. 59:11)

Kupersembahkan buku ini untuk Bapak, Ibu,
Mas Johan, Mas Hasan, Deffi Nuria
(Melatiku..Mawar Merahku..Sang Tambatan
Hati..Curahan Segenap Rasa) dan semua orang
yang telah membuatku berarti .....

ABSTRAK

ABSTRAK

Selama ini sistem enkripsi klasik yang ada tidak secara tuntas menyelesaikan masalah pengopian ilegal. Sekali enkripsi dihilangkan dari data rahasia, tidak ada lagi pengawasan pada penyebarannya. Tugas akhir ini bertujuan merancang dan membuat perangkat lunak untuk memproteksi data citra dengan metode *watermarking*. Data *watermark* disisipkan secara permanen ke dalam citra sehingga penerima citra yang mempunyai hak akses dapat dengan mudah melakukan akses ke citra. Modifikasi citra oleh *watermark* tidak menyebabkan penurunan kualitas citra secara drastis sehingga *watermark* yang disisipkan ke citra menjadi tidak tampak. *Watermark* tidak mungkin dihapus oleh penerima citra yang tidak mempunyai hak akses ke citra. *Watermark* adalah urutan tanda ASCII yang diperoleh melalui enkripsi dengan menggunakan kunci *private* dan kemudian diskalakan ke dalam bilangan real antara 0 sampai dengan 1. Caranya adalah dengan membagi tiap-tiap tanda ASCII dengan nilai 255.

Watermarking adalah metode yang beroperasi di dalam domain frekuensi, di mana urutan angka-angka real yang dihasilkan secara random disisipkan ke dalam kumpulan koefisien-koefisien DCT (*Discrete Cosine Transform*) yang dipilih.

Penyisipan *watermark* dilakukan pada koefisien transformasi dengan level di bawah *threshold* (nilai ambang) yang disesuaikan dengan probabilitas nilai koefisien transformasi untuk citra tertentu. IDEA (*International Data Encryption Algorithm*) adalah sebuah algoritma yang menggunakan sebuah *key* yang panjangnya 128-bit untuk melakukan enkripsi secara berturut-turut block-block 64-bit dari *plaintext*. *Plaintext* adalah bentuk asli dari sebuah pesan yang merupakan data input dari IDEA.

Kemampuan untuk memproteksi data sangat tergantung pada karakteristik dari sistem enkripsi. Sistem proteksi data dengan menggunakan metode *watermarking* ini, terdiri dari dua langkah penting. Langkah pertama adalah *watermark casting* di mana data *watermark* secara permanen disisipkan ke dalam citra. Langkah kedua adalah *watermark detection* yakni prosedur pendeteksian keberadaan data *watermark* di dalam citra.

KATA PENGANTAR

KATA PENGANTAR

Alhamdulillah, segala puja dan puji syukur ke hadirat Allah SWT, atas limpahan nikmat, rahmat dan hidayahNya, sehingga penulis dapat menyelesaikan tugas akhir dengan judul : **"Perancangan Dan Pembuatan Perangkat Lunak Proteksi Data Citra Dengan Menggunakan Metode *Watermarking*"**. Penulis sangat menyadari bahwa semua ini tidak lepas dari pertolongan dan kekuatan yang diberikan oleh Allah yang Maha Tinggi lagi Maha Agung.

Tugas akhir ini disusun guna memenuhi sebagian persyaratan untuk memperoleh gelar sarjana pada Jurusan Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember, Surabaya.

Berbagai hambatan dan tantangan penulis alami selama penyusunan tugas akhir ini. Tetapi berkat dorongan, semangat, motivasi dan bantuan yang diberikan berbagai pihak, akhirnya penulis dapat menyelesaikan tugas akhir ini. Untuk itu penulis menghaturkan terima kasih dan penghargaan sebesar-besarnya kepada :

1. **YANG MULIA AYAHANDA GURU MURSYID**, yang telah memberikan syafa'at dan pertolongannya kepada penulis.
2. Ibu Ir. Esther Hanaya, M.Sc., selaku dosen pembimbing pertama yang telah banyak memberikan bimbingan penulisan tugas akhir ini.
3. Bapak Rully Soelaiman, S.Kom., selaku dosen pembimbing kedua, yang telah memberikan ide, gagasan, saran, bimbingan dan dukungan bagi penulis, serta

ijin penggunaan fasilitas Laboratorium Sistem Informasi yang sangat menunjang terselesaikannya tugas akhir ini.

4. Bapak Dr. Ir. Arif Djunaidy, M.Sc., selaku ketua Jurusan Teknik Informatika ITS dan dosen wali selama masa perkuliahan, yang telah banyak memberikan arahan bagi penulis di dalam menuntut studi.
5. Seluruh Bapak dan Ibu Dosen di Jurusan Teknik Informatika ITS, atas ilmu dan bimbingannya selama masa perkuliahan.
6. Segenap staf dan karyawan Jurusan Teknik Informatika ITS, atas bantuannya dalam bidang administrasi kemahasiswaan.
7. Bapak, Ibu, Mas Johan, Mas Hasan, atas doa, restu dan segenap dorongan moril maupun materiil bagi penulis.
8. Pak Agus Shoheh Wibowo dan Bu Yayuk Suhartutik, atas doa, wejangan, serta pemberian perlengkapan ujian tugas akhir kepada penulis.
9. Pak Yarli, atas saran dan masukan pada presentasi penulis.
10. Sahabat-sahabatku, antara lain Ilham, Puguh, Bagus, Vita, Vefsi, Dea, Yusfita Evi Rosdiana, Dini, Titien, Dian N.R, Dimas, atas saran, kritik, *guyonan* dan "gangguan"-nya.
11. Juniorku, antara lain Salman, Eko, Anib, Darlis, Yuli Prapto, Yulius, Heri(98), Hera, Daning, yang senantiasa meramaikan lab, atas motivasi, kritik, "gangguan", serta dukungan fasilitas lab-nya.
12. Semua teman kos-kosan PK, antara lain Rosyid, Anang 'Bang Triman', Hartoyo 'HT', Abas 'Gibas', Ghozali 'Kakak', Hendro, Heru 'Siro', Danar,

Romli, Azis, Farkhan, Adib, Firman, Yuli, Warno, Koko, atas pinjaman buku, saran, motivasi, dan kritik bagi penulis.

13. Dewa, Padi, Sheila On 7 dan lain-lain, atas alunan nada-nada indahnya selama penulis mengerjakan tugas akhir.
14. Semua pihak yang telah membantu terselesaikannya tugas akhir ini.

Penulis menyadari, bahwa tugas akhir ini masih jauh dari sempurna, karenanya saran dan kritik yang membangun sangat diharapkan sebagai penyempurnaan.

Akhir kata, penulis berharap semoga penulisan tugas akhir ini dapat memberikan sumbangan yang bermanfaat bagi pembaca, sehingga hal ini bukan semata-mata sebagai tugas untuk menyelesaikan studi.

Surabaya, Agustus 2001

Penulis

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR TABEL

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dewasa ini marak sekali terjadi pembajakan-pembajakan terhadap suatu hasil karya cipta seseorang terutama foto-foto digital, yang sudah dipatenkan hak ciptanya. Kemudian hasil bajakan tersebut dijual ke konsumen dengan harga yang relatif jauh lebih murah dari harga yang telah ditentukan. Kejadian ini menunjukkan betapa mudahnya pengopian-pengopian ilegal terjadi, karena tidak adanya proteksi terhadap foto-foto digital tersebut. Hal ini menyebabkan penurunan kreativitas dan produktivitas para pencipta karya seni. Pelanggaran ini menunjukkan betapa hak cipta suatu hasil karya cipta seseorang tidak dihargai.

Berkembangnya teknologi komputer, terutama teknologi pengolahan citra digital, diharapkan memberikan solusi terhadap permasalahan ini. Oleh sebab itu dalam tugas akhir ini, penulis melakukan perancangan dan pembuatan perangkat lunak proteksi data citra dengan menggunakan metode *Watermarking*. Kelebihan dari *software* ini adalah data *watermark* yang disisipkan ke dalam citra dapat tidak menyebabkan penurunan kualitas citra (*image*). Bersifat permanen, sehingga sulit untuk dibajak.

1.2 Permasalahan

Kebanyakan perangkat lunak-perangkat lunak enkripsi yang ada tidak secara tuntas menyelesaikan masalah-masalah proteksi data citra, karena itu harus dicari cara yang efektif untuk mencegah "bahaya atau ancaman" tersebut. Atas dasar itulah penulis dalam tugas akhir ini menggunakan metode *Watermarking* sebagai alternatif yang efektif untuk mempertahankan keberadaan data *watermark* yang disisipkan ke dalam citra.

1.3 Tujuan

Penulis dalam tugas akhir ini menggunakan metode *Watermarking* sebagai alternatif yang efektif untuk mempertahankan keberadaan data *watermark* yang disisipkan di dalam citra.

Perangkat lunak yang akan dibuat dirancang untuk memproteksi data citra secara efektif untuk menjamin keberadaan data *watermark* yang disisipkan di dalam citra, dengan menggunakan metode *Watermarking*.

Watermarking adalah metode yang beroperasi di dalam domain frekuensi, di mana urutan angka-angka real yang dihasilkan secara random disisipkan ke dalam kumpulan koefisien-koefisien DCT yang dipilih. Metode ini tergolong baru dalam upaya memproteksi data citra, yang meliputi dua proses penting yaitu *watermark casting* dan *watermark detection*.

Proteksi data citra ini diarahkan untuk memperkenalkan bahwa data *watermark* yang disisipkan ke dalam citra, dapat tidak mempengaruhi kualitas citra secara drastis.

1.4 Batasan Masalah

Dalam tugas akhir ini permasalahan dibatasi sebagai berikut :

1. File gambar yang digunakan berbentuk *Bitmap* (BMP). File gambar ini adalah file yang dihasilkan oleh *Microsoft Windows* yang merupakan suatu perangkat lunak yang sudah memasyarakat di kalangan pengguna komputer desktop.
2. Citra yang digunakan sebagai input merupakan citra *grayscale*.
3. Ukuran citra maksimal 256 * 256 piksel. (Supaya proses komputasi citra tidak terlalu lama.)

1.5 Metodologi Tugas Akhir

Metodologi yang digunakan dalam pembuatan tugas akhir ini meliputi langkah-langkah sebagai berikut :

- Pengumpulan data (studi literatur)

  Pada tahap ini dilakukan studi literatur terhadap konsep-konsep atau teori yang berkaitan dengan tugas akhir ini, termasuk juga berdiskusi dengan dosen pembimbing, dosen dan rekan kuliah.

- Perancangan perangkat lunak

  Pada tahap ini dilakukan perancangan struktur data, algoritma, dan diagram alur yang akan digunakan untuk implementasi dalam perangkat lunak yang akan dibuat.

- Pembuatan perangkat lunak

  Pada tahap ini dilakukan pengimplementasian struktur data dan algoritma yang telah dirancang sebelumnya ke dalam bahasa pemrograman.

- Pengujian dan evaluasi perangkat lunak

  Pada tahap ini dilakukan pengujian perangkat lunak yang telah dibuat, pengevaluasian terhadap hasil yang diperoleh dan perbaikan (revisi) program bilamana hasil belum sesuai dengan tujuan yang diharapkan.

- Penyusunan naskah tugas akhir (dokumentasi)

  Pada tahap ini dilakukan penyusunan dokumentasi agar dapat dipelajari untuk pengembangan lebih lanjut, diantaranya akan menjelaskan dasar teori yang digunakan dan metode yang terlibat di dalamnya, desain perangkat lunak dan

implementasinya, serta hasil pengujian sistem. Termasuk juga perbaikan kesalahan-kesalahan yang masih terdapat dalam laporan, baik struktur penulisan, tata tulis, ejaan dan kekurangan-kekurangan lainnya.

1.6 Sistematika Pembahasan

Dalam tugas akhir ini sistematika pembahasan yang diterapkan adalah sebagai berikut :

BAB I **PENDAHULUAN,** menjelaskan tentang hal-hal yang mendasar serta memotivasi perancangan dan pembuatan perangkat lunak, meliputi latar belakang, permasalahan, tujuan, batasan masalah, metodologi tugas akhir, dan sistematika pembahasan.

BAB II **ENKRIPSI CITRA,** berisikan Enkripsi secara umum, istilah-istilah dalam Enkripsi dan menjelaskan dua bentuk umum algoritma Enkripsi berdasarkan penggunaan *key* serta IDEA sebagai proses enkripsi, algoritma *Euclid's* dan *Extended Euclid's* merupakan algoritma pendukung IDEA.

BAB III **TRANSFORMASI COSINUS DISKRIT,** mengulas secara lengkap tentang Transformasi Cosinus Diskrit. Rumus-rumus yang digunakan, dan penggunaannya untuk mengubah nilai piksel ke nilai frekuensi.

BAB IV METODE WATERMARKING, mengulas dasar digunakannya *Watermarking* sebagai metode dalam menyelesaikan proteksi data citra. Rumus-rumus yang digunakan, dan penggunaannya untuk menyelesaikan masalah ini.

BAB V PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK, menjelaskan tentang pemilihan struktur data dan implementasi perangkat lunak.

BAB VI UJI COBA DAN EVALUASI, membahas hasil uji coba perangkat lunak yang sudah jadi, dan menganalisa kehandalan sistem proteksi data citra yang dihasilkan.

BAB VII KESIMPULAN DAN SARAN, menguraikan kesimpulan dari bab-bab sebelumnya serta saran yang berkaitan dengan arah pengembangan perangkat lunak selanjutnya.

BAB II

ENKRIPSI CITRA

BAB II

ENKRIPSI CITRA

2.1 Enkripsi dan Dekripsi

Enkripsi adalah sebuah proses menyandikan sebuah pesan sebagai upaya untuk menyembunyikan substansinya. Enkripsi juga berarti menjamin keutuhan atau keaslian data di dalam sebuah lingkungan yang menyebabkan data mengalami kerusakan atau data menjadi tidak valid. Dekripsi adalah proses kebalikan, perubahan bentuk sebuah pesan enkripsi kembali ke dalam bentuk asalnya. Sebuah sistem untuk enkripsi dan dekripsi guna mendapatkan bentuk asli dari sebuah pesan disebut *Cryptosystem*. *Cryptosystem* ditunjukkan sebagai $P = D(E(P))$. Bentuk asli dari sebuah pesan disebut dengan *plaintext* atau *cleartext*, merupakan data input dari IDEA. IDEA adalah sebuah block *cipher*, yang beroperasi pada block *plaintext* dan *ciphertext* 64-bit yang dikendalikan oleh *key* 128-bit. Menggunakan *subkey* yang berbeda, proses enkripsi terdiri dari delapan langkah enkripsi yang sama yang diikuti oleh sebuah output transformasi. Output dari IDEA disebut *ciphertext*. Situasi ini ditunjukkan dalam Gambar 2.4.

Sebuah pesan *plaintext* P merupakan rangkaian dari karakter-karakter $P = (p_1, p_2, \ldots, p_n)$, *ciphertext* juga merupakan rangkaian dari karakter-karakter

$C = (c_1, c_2, \ldots, c_m)$, himpunan c_i bisa sama dengan himpunan p_i, karena panjang *ciphertext* dan *plaintext* sama-sama 64 bit. Perubahan bentuk antara *plaintext* dan *ciphertext* ditunjukkan sebagai $C = E(P)$ dan $P = D(C)$. C menggambarkan *ciphertext*, yang kadang-kadang berukuran sama seperti P tetapi kadang-kadang lebih besar. E adalah algoritma enkripsi, P adalah *plaintext*, dan D adalah algoritma dekripsi.

Gambar 2.1 Enkripsi dan Dekripsi

2.2 Algoritma Enkripsi dengan Key

Semua algoritma enkripsi modern menggunakan sebuah *key*, dinyatakan dengan k. *Key* ini dapat diambil dari himpunan nilai (sebuah bilangan yang besar adalah lebih baik). Nilai dari *key* mempengaruhi fungsi-fungsi enkripsi dan dekripsi, sehingga fungsi-fungsi enkripsi dan dekripsi sekarang menjadi :

$$E_k(P) = C$$
$$D_k(C) = P$$

dan jika *key* enkripsi dan *key* dekripsi adalah sama, maka :

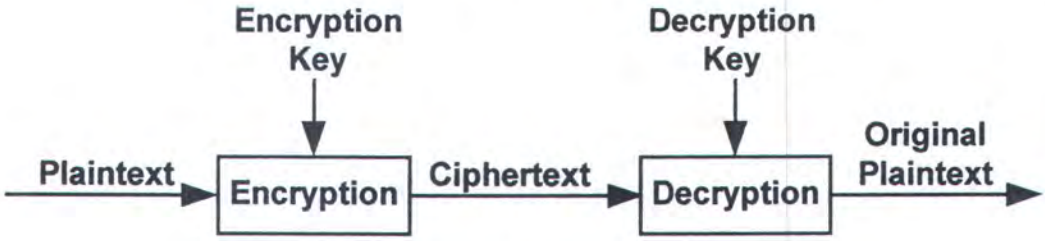
$$D_k(E_k(P)) = P$$

Hal ini ditunjukkan dalam Gambar 2.2.

Gambar 2.2 Enkripsi dan Dekripsi dengan Key

Terdapat algoritma di mana *key* enkripsi dan *key* dekripsi merupakan pasangan nilai, seperti yang ditunjukkan dalam Gambar 2.3. Jadi, *key* enkripsi k_1 berbeda dari *key* dekripsi k_2, seperti berikut ini :

$$E_{k_1}(P) = C$$
$$D_{k_2}(C) = P$$
$$D_{k_2}\left(E_{k_1}(P)\right) = P$$



Gambar 2.3 Enkripsi dan Dekripsi dengan dua Key

Ada dua bentuk umum dari algoritma-algoritma yang didasarkan pada *key* yaitu : *Symmetric* dan *Public-key*. Algoritma *Symmetric* adalah algoritma di mana *key* enkripsi dapat dihitung dari *key* dekripsi dan sebaliknya. Pada umumnya sistem menggunakan *key* enkripsi dan *key* dekripsi yang sama. Algoritma ini, juga disebut *secret-key algorithms*, *single-key algorithms*, atau *one-key algorithms*, mengharuskan pengirim dan penerima untuk menyetujui sebuah *key* sebelum mereka saling menyampaikan pesan-pesan. *Key* ini harus dijaga kerahasiaannya.

Perlindungan algoritma *Symmetric* terletak di dalam *key*, membuka rahasia *key* berarti semua orang akan dapat melakukan enkripsi dan dekripsi pesan di dalam *cryptosystem* ini.

Enkripsi dan dekripsi dengan algoritma *Symmetric* dinyatakan dengan notasi matematika :

$$E_k(P) = C$$
$$D_k(C) = P$$

Algoritma *Symmetric* dapat di bagi ke dalam dua kategori. Yang pertama beroperasi dengan *plaintext* yang terdiri dari bit tunggal di setiap saat, dan ini disebut *stream algorithms* atau *stream ciphers*. Sedangkan yang lain beroperasi dengan *plaintext* di dalam kelompok bit. Kelompok bit disebut block, dan algoritmanya disebut *block algorithms* atau *block ciphers*. Untuk algoritma-algoritma yang terimplementasikan di komputer, digunakan sebuah block khusus berukuran 64-bit. Di dalam kedua algoritma *block* dan *stream, key* yang sama digunakan untuk enkripsi dan dekripsi.

Algoritma *Public-key* adalah berbeda. Algoritma *Public-key* dirancang sedemikian hingga *key* yang digunakan untuk enkripsi berbeda dengan *key* yang digunakan untuk dekripsi. Lebih lanjut, *key* dekripsi tidak dapat dihitung dari *key* enkripsi. Mereka disebut sebagai sistem *public-key* karena *key* enkripsi dapat dibuat publik. Semua orang yang tak dikenal dapat menggunakan *key* enkripsi untuk melakukan enkripsi sebuah pesan, tetapi hanya seseorang dengan *key* dekripsi yang bersesuaian dapat melakukan dekripsi sebuah pesan. Dalam sistem

ini, *key* enkripsi sering disebut *public-key*, dan *key* dekripsi sering disebut *private-key*. Enkripsi menggunakan *public-key* k dinyatakan dengan :

$$E_k(P) = C$$

Meskipun *public-key* dan *private-key* adalah berbeda, dekripsi dengan *private-key* yang bersesuaian dinyatakan dengan :

$$D_k(C) = P$$

Kadang-kadang, pesan dienkrip dengan *private-key* dan didekrip dengan *public-key*.

$$E_k(P) = C$$
$$D_k(C) = P$$

2.3 IDEA

IDEA termasuk dalam kategori Algoritma *Public-key*, karena *key* yang digunakan untuk enkripsi berbeda dengan *key* yang digunakan untuk dekripsi. IDEA adalah sebuah block *cipher* yang menggunakan sebuah *key* yang panjangnya 128-bit untuk melakukan enkripsi secara berturut-turut block-block 64-bit dari *plaintext*. Algoritma ini dirancang dengan menggunakan *subkey* yang dihasilkan dari *key* dengan serangkaian operasi-operasi sebagai berikut :

XOR

Penjumlahan modulo 2^{16} (mengabaikan ***overflow***)

Perkalian modulo $2^{16}+1$ (mengabaikan ***overflow***)

pada segmen-segmen dari block *plaintext* 64-bit. Susunan enkripsi menggunakan sebanyak 52 *subkey* 16-bit yang dihasilkan dari *key* 128-bit sebagai berikut :

key 128-bit dibagi menjadi delapan *key* 16-bit, yang merupakan delapan *subkey* pertama. Bit-bit dari *key* 128-bit dirotasi 25 bit ke kiri untuk membuat sebuah *key* baru yang dibagi menjadi delapan *subkey* 16-bit berikutnya. Langkah kedua diulang sampai dihasilkan lima puluh dua *subkey*. Enkripsi meliputi perkalian modular dengan modulus dari $2^{16}+1$ dan penjumlahan modular dengan modulus dari 2^{16}. Block *plaintext* 64-bit dibagi menjadi empat segmen 16-bit yang disebut dengan P1, P2, P3 dan P4 dengan *subkey-subkey* S1, S2, S3, S4.....S52.

Proses enkripsi terdiri dari delapan langkah enkripsi yang sama (diketahui sebagai putaran enkripsi) yang diikuti oleh sebuah *output* transformasi. Setiap putaran meliputi langkah-langkah sebagai berikut :

P1 * S1 → D1

P2 + S2 → D2

P3 + S3 → D3

P4 * S4 → D4

D1 **XOR** D4 → D5

D2 **XOR** D4 → D6

D5 * S5 → D7

D6 + D7 → D8

D8 * S6 → D9

D7 + D9 → D10

D1 **XOR** D9 → D11

D3 **XOR** D9 → D12

D2 **XOR** D10 → D13

D4 **XOR** D10 → D14

Setelah proses ini block-block D12, D13 ditukar sehingga menjadi D11, D13, D12 dan D14 yang digunakan sebagai input untuk putaran berikutnya bersama-sama dengan enam *subkey* berikutnya, S7 sampai dengan S12. Prosedur ini diikutkan untuk delapan putaran yang memberi sebanyak empat *output* block yang disebut E1, E2, E3 dan E4. Empat langkah berikutnya menggunakan empat *subkey* yang terakhir untuk melengkapi enkripsi :

E1 * S49 → C1

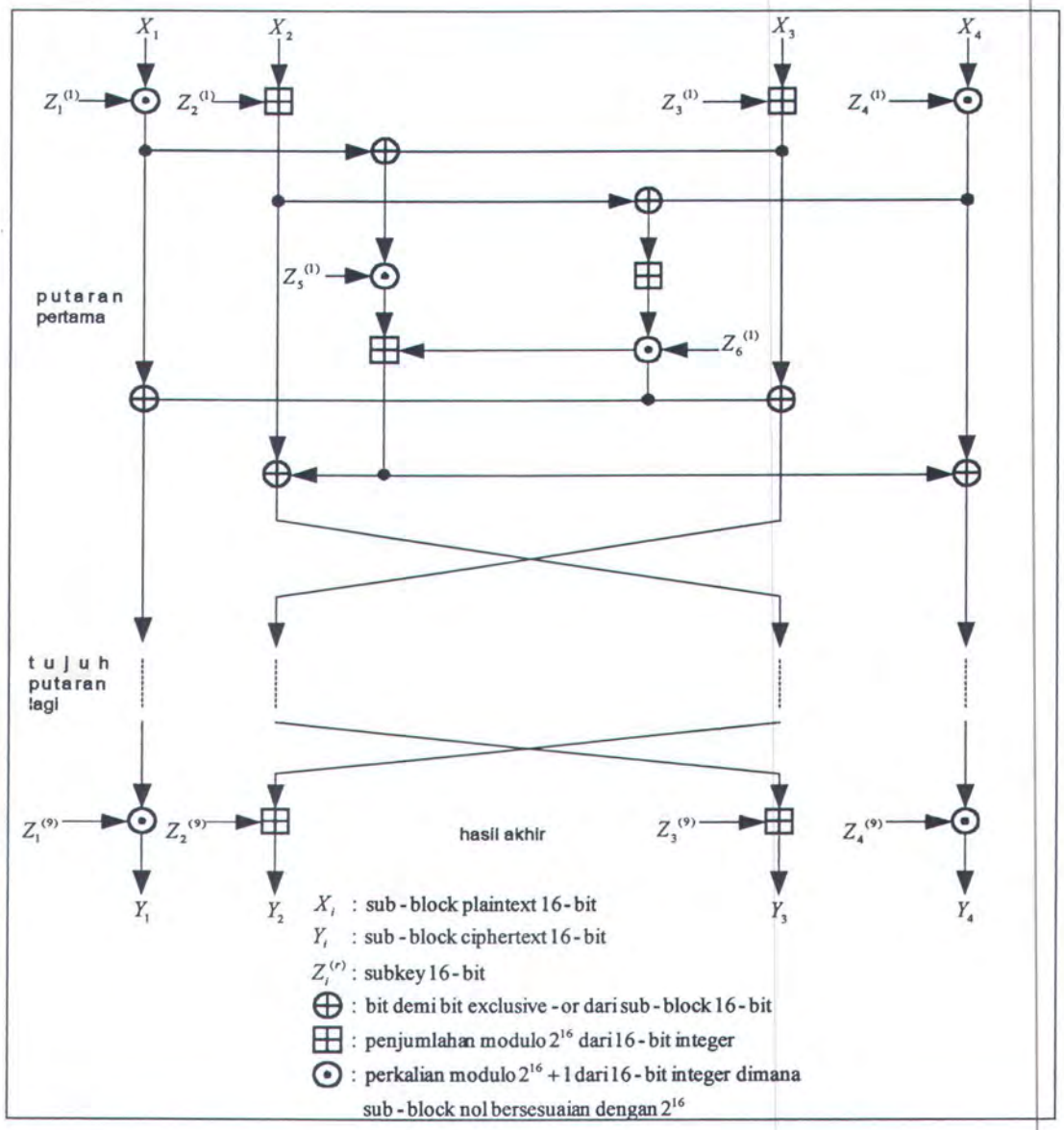
E2 + S50 → C2

E3 + S51 → C3

E4 * S52 → C4

Sebuah *key* yang bit-bit nya berisi nol semua didefinisikan sama dengan 2^{16} untuk langkah-langkah modular perkalian. Empat *output* block yang terakhir, C1 sampai C4, dirangkai kembali membentuk sebuah block 64-bit dari *ciphertext*. Keseluruhan proses diulang untuk block-block 64-bit dari *plaintext* secara berturut-turut sampai seluruh *plaintext* telah dienkripsi. Dekripsi juga menggunakan urutan operasi-operasi yang sama berturut-turut block 64-bit dari *ciphertext*, tetapi dengan sekumpulan *subkey* yang berbeda. *Subkey-subkey* dekripsi disusun dari *subkey-subkey* enkripsi yang didapat melalui salah satu invers perkalian atau penjumlahan dari prosedure yang menghasilkan *subkey-subkey* dekripsi. *Subkey-subkey* dekripsi berhubungan dengan *subkey-subkey* enkripsi dari S1 sampai dengan S52.

Proses yang lain, setiap putaran enkripsi menghasilkan enam *subkey* 16-bit dari *key* 128-bit. Selanjutnya empat *subkey* 16-bit yang berikutnya digunakan sebagai *output* transformasi, ada 52 (= 8 * 6 + 4) *subkey* 16-bit yang berbeda telah dihasilkan dari *key* 128-bit. Di dalam putaran enkripsi pertama, empat *subkey* 16-bit pertama bergabung dengan dua block *plaintext* 16-bit menggunakan penjumlahan modulo 2^{16}, dan dengan dua block *plaintext* 16-bit yang lain menggunakan perkalian modulo $2^{16}+1$. Hasilnya kemudian di proses lebih lanjut. Dua *subkey* 16-bit selebihnya masuk dalam perhitungan dan operator aljabar grup yang ketiga, digunakan untuk XOR bit demi bit. Di akhir dari putaran enkripsi pertama dihasilkan empat nilai 16-bit yang digunakan sebagai input untuk putaran enkripsi kedua di dalam sebagian urutan yang dirubah. Proses yang digambarkan di atas untuk putaran satu diulang di dalam setiap putaran berikutnya dari tujuh putaran enkripsi yang menggunakan *subkey* 16-bit yang berbeda untuk setiap kombinasi. Pada waktu *output* transformasi berikutnya, empat nilai 16-bit yang dihasilkan di akhir putaran enkripsi ke delapan bergabung dengan empat yang terakhir dari 52 *subkey* menggunakan penjumlahan modulo 2^{16} dan perkalian modulo $2^{16}+1$ ke bentuk yang menghasilkan empat block *ciphertext* 16-bit.



Gambar 2.4 Sketsa dari IDEA

PUTARAN	SUBKEY ENKRIPSI	SUBKEY DEKRIPSI
1 :	$Z_1^{(1)}\ Z_2^{(1)}\ Z_3^{(1)}\ Z_4^{(1)}\ Z_5^{(1)}\ Z_6^{(1)}$	$Z_1^{(9)}$ -1 -$Z_2^{(9)}$ -$Z_3^{(9)}$ $Z_4^{(9)}$ -1 $Z_5^{(8)}\ Z_6^{(8)}$
2 :	$Z_1^{(2)}\ Z_2^{(2)}\ Z_3^{(2)}\ Z_4^{(2)}\ Z_5^{(2)}\ Z_6^{(2)}$	$Z_1^{(8)}$ -1 -$Z_2^{(8)}$ -$Z_3^{(8)}$ $Z_4^{(8)}$ -1 $Z_5^{(7)}\ Z_6^{(7)}$
3 :	$Z_1^{(3)}\ Z_2^{(3)}\ Z_3^{(3)}\ Z_4^{(3)}\ Z_5^{(3)}\ Z_6^{(3)}$	$Z_1^{(7)}$ -1 -$Z_2^{(7)}$ -$Z_3^{(7)}$ $Z_4^{(7)}$ -1 $Z_5^{(6)}\ Z_6^{(6)}$
4 :	$Z_1^{(4)}\ Z_2^{(4)}\ Z_3^{(4)}\ Z_4^{(4)}\ Z_5^{(4)}\ Z_6^{(4)}$	$Z_1^{(6)}$ -1 -$Z_2^{(6)}$ -$Z_3^{(6)}$ $Z_4^{(6)}$ -1 $Z_5^{(5)}\ Z_6^{(5)}$
5 :	$Z_1^{(5)}\ Z_2^{(5)}\ Z_3^{(5)}\ Z_4^{(5)}\ Z_5^{(5)}\ Z_6^{(5)}$	$Z_1^{(5)}$ -1 -$Z_2^{(5)}$ -$Z_3^{(5)}$ $Z_4^{(5)}$ -1 $Z_5^{(4)}\ Z_6^{(4)}$
6 :	$Z_1^{(6)}\ Z_2^{(6)}\ Z_3^{(6)}\ Z_4^{(6)}\ Z_5^{(6)}\ Z_6^{(6)}$	$Z_1^{(4)}$ -1 -$Z_2^{(4)}$ -$Z_3^{(4)}$ $Z_4^{(4)}$ -1 $Z_5^{(3)}\ Z_6^{(3)}$
7 :	$Z_1^{(7)}\ Z_2^{(7)}\ Z_3^{(7)}\ Z_4^{(7)}\ Z_5^{(7)}\ Z_6^{(7)}$	$Z_1^{(3)}$ -1 -$Z_2^{(3)}$ -$Z_3^{(3)}$ $Z_4^{(3)}$ -1 $Z_5^{(2)}\ Z_6^{(2)}$
8 :	$Z_1^{(8)}\ Z_2^{(8)}\ Z_3^{(8)}\ Z_4^{(8)}\ Z_5^{(8)}\ Z_6^{(8)}$	$Z_1^{(2)}$ -1 -$Z_2^{(2)}$ -$Z_3^{(2)}$ $Z_4^{(2)}$ -1 $Z_5^{(1)}\ Z_6^{(1)}$
Output Transformasi	$Z_1^{(9)}\ Z_2^{(9)}\ Z_3^{(9)}\ Z_4^{(9)}$	$Z_1^{(1)}$ -1 -$Z_2^{(1)}$ -$Z_3^{(1)}$ $Z_4^{(1)}$ -1

Tabel 2.1 Subkey Enkripsi dan Dekripsi

2.4 Algoritma Euclid's

Algoritma *Euclid's* digunakan untuk menemukan GCD (*Greatest Common Divisor*) dari dua bilangan **a** dan **n,** di mana **a < n.**

GCD (**a**,**n**) diberikan sebagai berikut :

1. $g_0 = n$
2. $g_1 = a$
3. $g_{i+1} = g_{i-1} \mod g_i$, untuk $(i >= 1)$
4. Ulangi langkah 3 sampai g indek ke-i sama dengan nol
5. Jika $g_i = 0$ maka GCD (**a**,**n**) = g_{i-1}

2.5 Algoritma Extended Euclid's

Dalam algoritma Extended Euclid's untuk menghitung invers, bilangan yang digunakan sebagai pembagi harus bilangan integer yang berkisar antara 0 sampai suatu bilangan **n**. Apabila nilai hasil operasi-operasi perkalian dan penjumlahan dengan modulo **n** di luar *range* $0,\ldots n-1$, maka nilai tersebut sama dengan sisa pembagian nilai tersebut dengan **n**. (Alternatif lain, menjumlahkan atau mengurangi nilai tersebut dengan **n** sampai nilai diantara 0 dan **n**-1.)

Contoh

Operasi penjumlahan : **n** = 13
t = 8 + 7 = 15
sisa = **t** mod **n** = 2

Operasi perkalian : **n** = 13
t = 4 * 5 = 20
sisa = **t** mod **n** = 7

Analogi dari pembagian adalah perkalian dengan invers dari sebuah bilangan.

Contoh

Invers dari 3 modulo 5 adalah 2, karena 3 * 2 = 6 mod 5 = 1.

Supaya suatu bilangan mempunyai invers, maka **n** harus merupakan bilangan prima. Jika **n** merupakan bilangan prima, maka GCD (**a**,**n**) selalu bernilai 1, sehingga setiap bilangan kecuali 0 mempunyai invers. (0 tidak pernah mempunyai invers, sebab 0 dikalikan dengan bilangan berapapun tidak akan bernilai 1.)

Contoh

Setiap bilangan a diantara 1 sampai dengan **n**-1 mempunyai invers a^{-1}, sebab $a * a^{-1}$ mod **n** = 1.

- **Algoritma *Euclid's*** dapat diperluas untuk menemukan invers dengan iterasi dari $g_i = u_i * n + v_i * a$
- **Algoritma *Extended Euclid's*** (atau Binary GCD) untuk menemukan invers dari sebuah bilangan **a** mod **n** (invers (**a**,**n**)), di mana GCD (**a**,**n**) = 1 adalah sebagai berikut :

1. $g_0 = n$
2. $u_0 = 1$
3. $v_0 = 0$
4. $g_1 = a$
5. $u_1 = 0$
6. $v_1 = 1$
7. $y = g_{i-1}\ div\ g_i$

   $g_{i+1} = g_{i-1} - y * g_i = g_{i-1} \bmod g_i$

   $u_{i+1} = u_{i-1} - y * u_i$

   $v_{i+1} = v_{i-1} - y * v_i$

   untuk $(i >= 1)$
8. Ulangi langkah 7 sampai g indek ke-i sama dengan nol
9. Jika $g_i = 0$ maka invers (**a**,**n**) = v_{i-1}

BAB III

TRANSFORMASI COSINUS DISKRIT

BAB III

TRANSFORMASI COSINUS DISKRIT

Teknik *Image Watermarking* sejauh ini dibagi ke dalam dua kelompok utama. Yang pertama, *watermark* disisipkan secara langsung ke dalam *domain spasial*. Sedangkan teknik yang lain, *watermark* disisipkan ke dalam *domain transformasi*.

Dalam tugas akhir ini penulis menggunakan kelompok yang kedua yaitu *watermark* disisipkan ke dalam *domain transformasi*.

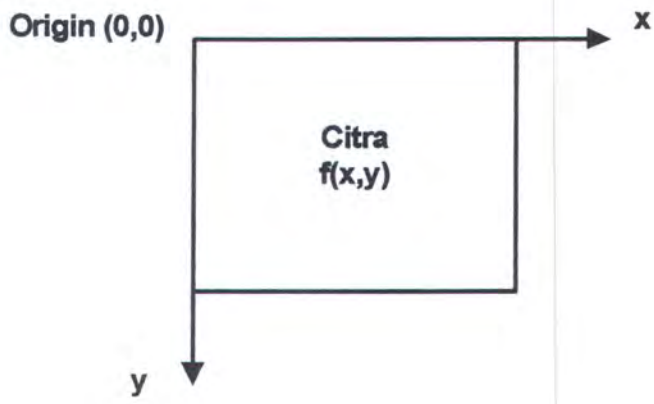
Gambar 3.1 Tiga Tahapan Operasi Dalam Teknik Watermarking

Transformasi dapat diberlakukan pada keseluruhan citra atau subbagian (block) dari citra. Untuk menyisipkan *watermark* ke dalam citra, dipilih beberapa koefisien di dalam *domain transformasi*, kemudian dimodifikasi menurut aturan *Watermarking*. Koefisien-koefisien dapat dimodifikasi pada keseluruhan citra atau pada beberapa koefisien terpilih saja. Pada *watermark detection*, terdapat proses penghitungan terhadap korelasi dan threshold. Kemudian membandingkannya untuk mendeteksi keberadaan data *watermark* di dalam citra.

3.1 Pemodelan Citra

Citra merupakan matriks dua dimensi dari fungsi intensitas cahaya. Karena itu referensi citra menggunakan dua variabel yang menunjuk posisi pada bidang dengan sebuah fungsi intensitas cahaya yang dapat dituliskan sebagai $f(x,y)$ dengan f adalah nilai amplitudo pada koordinat spasial (x,y). Karena cahaya merupakan salah satu bentuk energi, $f(x,y)$ tak berharga nol atau negatif dan merupakan bilangan berhingga atau jika dinyatakan dalam bentuk persamaan matematis dapat ditunjukkan seperti persamaan (3.1). Sedangkan konversi sistem koordinat citra diskrit ditunjukkan dalam gambar 3.2.

$$0 < f(x,y) < \infty \tag{3.1}$$



Gambar 3.2 Konversi sistem koordinat citra diskrit

Citra yang kita lihat sehari-hari merupakan cahaya yang direfleksikan sebuah objek. Fungsi $f(x,y)$ dapat dilihat sebagai fungsi dengan dua unsur. Unsur yang pertama merupakan kekuatan sumber cahaya yang melingkupi pandangan kita

terhadap objek (*illumination*). Unsur yang kedua merupakan besarnya cahaya yang direfleksikan oleh objek ke dalam pandangan kita (*reflectance components*). Keduanya dituliskan sebagai fungsi $i(x,y)$ dan $r(x,y)$. Fungsi $f(x,y)$ dapat dituliskan dengan persamaan :

$$f(x,y) = i(x,y)\ r(x,y) \tag{3.2}$$

di mana

$0 < i(x,y) < \infty$ dan $0 < r(x,y) < 1$

Citra digital (*digital image*) adalah citra kontinyu $f(x,y)$ yang sudah didiskritkan, baik koordinat spasial maupun tingkat kecerahannya. Kata "kontinyu" di sini menjelaskan bahwa indeks x dan y dapat bernilai pecahan, sedangkan kata diskrit menjelaskan bahwa indeks x dan y hanya bernilai bulat. Citra digital (berikutnya akan disingkat dengan citra) dapat dianggap sebagai matriks dengan ukuran $M \times N$ yang baris dan kolomnya menunjukkan titik-titiknya, yang diperlihatkan pada persamaan (3.3), sedangkan nilai dari elemen matriks merepresentasikan tingkat keabuan atau warna pada titik tersebut. Elemen dari matriks dua dimensi ini disebut dengan elemen citra (*image element*), pel atau piksel.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{3.3}$$

3.2 Transformasi Cosinus Diskrit [3]

Langkah pertama operasi dalam teknik *Watermarking* adalah transformasi matematik yang dikenal dengan nama *Discrete Cosine Transform* disingkat DCT. DCT dalam kelas operasi matematika termasuk dalam FFT (*Fast Fourier Transform*). Transformasi adalah mekanisme pengubahan data yang ada ke bentuk data yang lain, baik kuantitas dan kualitasnya, sehingga mempermudah kita dalam melakukan analisis terhadap data tersebut. Transformasi *Fourier* merupakan metode yang sangat berguna dalam pengolahan sinyal.

Transformasi *Fourier* dikerjakan selama menganalisa citra dengan menggunakan FFT. Pengumpulan titik-titik sampel dari sebuah sinyal citra menghasilkan himpunan titik dengan domain waktu. Titik-titik tersebut menghubungkan dimensi waktu pada sumbu horisontal dengan level tegangan di sumbu vertikal. FFT mentransformasikan titik tersebut ke himpunan nilai frekuensi yang mewakili sinyal yang ekivalen.

Hal penting lainnya tentang transformasi ini adalah sifatnya yang *reversible* (dapat dibalik).

DCT adalah transformasi yang sangat mirip dengan transformasi *Fourier* dan menghasilkan produk yang sejenis. Transformasi ini akan mengolah titik-titik dari *domain spasial* dan mentransformasikannya ke bentuk sejenis dalam *domain frekuensi.* DCT mengolah sinyal tiga dimensi yang digambar pada sumbu-sumbu X, Y, dan Z.

---

[3] I Nyoman Suyoga, "Rancang Bangun Perangkat Lunak Kompresi Dan Dekompresi Dengan Menggunakan Metode Transformasi Cosinus Diskrit",Teknik Informatika, ITS di Surabaya,1995

Pada kasus ini sinyal adalah gambar grafik. Sumbu-sumbu X dan Y menunjukkan frekuensi dari sinyal dalam dua dimensi yang berbeda. Amplitudo dari sinyal pada kasus ini adalah nilai dari piksel pada titik di layar. Sebagai contoh adalah nilai yang digunakan untuk menyajikan gambar dalam derajat keabuan (*grey-scale*). Gambar grafik yang ditampilkan di layar dapat dianggap sebagai sinyal tiga dimensi yang kompleks dengan nilai sumbu Z ditunjukkan oleh warna pada layar pada titik yang bersangkutan. Ini adalah sinyal yang direpresentasikan pada *domain spasial*. DCT dapat digunakan untuk mengubah informasi spasial ke bentuk frekuensi atau spektral. Dan seperti FFT, terdapat fungsi invers DCT (IDCT) yang dapat membalikkan representasi spektral ke bentuk spasial mula-mula.

3.3 Penurunan Rumus untuk Matriks DCT

Rumus untuk DCT dua dimensi dan rumus untuk inversnya IDCT adalah sebagai berikut :

$$DCT(i,j) = \frac{2}{N} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) Cos\left[\frac{(2x+1)i\pi}{2N}\right] Cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{untuk } x = 0 \\ 1 & \text{untuk } x > 0 \end{cases}$$

i = posisi baris untuk koefisien DCT
j = posisi kolom untuk koefisien DCT
x = posisi baris untuk piksel
y = posisi kolom untuk piksel
N = banyaknya baris atau kolom

Gambar 3.3 Rumus Transformasi Cosinus Diskrit

DCT ditunjukkan dengan sebuah matrik bujur sangkar $N \times N$ dari nilai-nilai piksel, dan akan menghasilkan sebuah matrik bujur sangkar $N \times N$ dari koefisien frekuensi.

$$Pixel(x,y) = \frac{2}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} C(i)\,C(j)\,DCT(i,j)\,Cos\left[\frac{(2x+1)i\pi}{2N}\right]Cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{untuk} \quad x = 0 \\ 1 & \text{untuk} \quad x > 0 \end{cases}$$

i = posisi baris untuk koefisien DCT
j = posisi kolom untuk koefisien DCT
x = posisi baris untuk piksel
y = posisi kolom untuk piksel
N = banyaknya baris atau kolom

Gambar 3.4 Rumus Invers Transformasi Cosinus Diskrit

Piksel ditunjukkan dengan sebuah matrik bujur sangkar $N \times N$ dari koefisien-koefisien DCT, dan akan menghasilkan sebuah matrik bujur sangkar $N \times N$ dari nilai-nilai piksel.

Untuk menulis program DCT dan IDCT. Fungsi *cosinus* yang harus dikalikan bersama-sama hanya perlu dikalikan pada awal program. Dan hasilnya disimpan di dalam array satu dimensi untuk penggunaan berikutnya.

Dalam matrik DCT $N \times N$, semua elemen dalam baris ke 0 memiliki komponen frekuensi nol pada arah yang sama. Semua elemen dalam kolom ke 0 memiliki komponen frekuensi nol dalam arah yang lain. Sejalan dengan bergesernya baris dan kolom dari titik asal (0,0) , koefisien-koefisien dalam matrik DCT yang ditransformasikan mulai menunjukkan kenaikan frekuensi, dengan frekuensi tertinggi pada koordinat ($N-1$, $N-1$).

Ini sangatlah berarti, karena gambar grafis pada layar dibentuk dari informasi yang berfrekuensi rendah. Sejalan dengan bergesernya baris dan kolom

dari titik asal (0,0), koefisien-koefisien tidak saja cenderung memiliki nilai yang lebih rendah melainkan mereka juga membawa informasi yang tidak sepenting koefisien-koefisien sebelumnya.

Maka transformasi DCT menunjukkan informasi dalam sinyal yang dapat diabaikan tanpa menimbulkan dampak yang serius pada kualitas gambar. Sulit dibayangkan bagaimana caranya menentukan informasi yang dapat diabaikan pada gambar yang tidak ditransformasikan. Jika gambarnya masih dalam *domain spasial* yang menggunakan piksel-piksel, program akan sulit menggambarkan piksel-piksel mana yang penting dan yang mana yang tidak.

3.4 Threshold Selection [2]

Setelah mendefinisikan hasil transformasi DCT berupa urutan data $X(m), m = 0,1,\ldots,(M-1)$, maka salah satu sifat transformasi citra yang akan dimanfaatkan di sini adalah pemampatan energi pada frekuensi rendah. Dengan mengikuti aturan *threshold coding* maka dapat ditentukan koefisien yang akan digantikan oleh *watermark* tersebut.

Dalam *threshold coding*, koefisien transformasi dibandingkan dengan *threshold*, kemudian koefisien yang nilainya lebih kecil daripada *threshold* akan dikodekan. lokasi dari koefisien yang dikodekan tidak diketahui sebelumnya, dan informasi lokasi harus dikodekan pada *threshold coding*. Alasan-alasan inilah yang mendasari pemakaian sistem *watermark* berbasis *threshold coding*.

---

[2] A. Piva, M. Barni, F. Bartolini, V. Cappellini. "Threshold Selection for Correlation-Based Watermark Detection", Dipartimento di Ingegneria Elettronica, Universita di Firenze

$$\overline{\sigma_v^{\,2}} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}\left[v_i^{\,2}\right] \tag{3.1}$$

$i = 1, 2, 3, \ldots, n$

n = panjang dari watermark

v_i = koefisien - koefisien yang ditandai

$\sigma_v^{\,2}$ = variansi

$\overline{\sigma_v^{\,2}}$ = nilai rata - rata dari variansi

$$T_\rho = 3.3\sqrt{\frac{2\left(1+\alpha^2\right)\overline{\sigma_v^{\,2}}}{n}} \tag{3.2}$$

α = energi watermark

T_ρ = threshold watermark

Nilai untuk (α) energi *watermark* ditentukan melalui dialog input.

BAB IV

METODE WATERMARKING

BAB IV

METODE WATERMARKING

Enkripsi *plaintext* dengan menggunakan kunci *private* akan menghasilkan urutan tanda ASCII yang kemudian diskalakan ke dalam bilangan real antara 0 sampai dengan 1. Caranya adalah dengan membagi tiap-tiap tanda ASCII dengan nilai 255 dan kemudian menyisipkannya ke dalam citra sebagai data *watermark*. Penyisipan *watermark* pada citra diakui merupakan cara yang efektif untuk melindungi citra dari pembajakan. Data *watermark* disisipkan ke dalam *domain* transformasi yang terpilih dan dimodifikasi menurut aturan *watermarking*, hal ini menyebabkan data *watermark* menjadi tidak tampak dan bersifat permanen di dalam citra sehingga dapat digunakan untuk menyimpan identitas pembuat atau pemilik. Berbagai macam algoritma telah dicoba untuk menyembunyikan informasi dalam *domain spasial* atau dalam *domain* hasil transformasi, agar tidak mudah dirusak.

IDEA digunakan untuk enkripsi *plaintext*, karena IDEA lebih cepat jika dibandingkan dengan DES.

Penyisipan *watermark* dilakukan pada koefisien transformasi dengan level di bawah *threshold* yang disesuaikan dengan probabilitas nilai koefisien transformasi untuk citra tertentu.

Pada kasus ini[6] *watermark* akan berupa sample data bilangan real $\{x_1, x_2 \ldots x_n\}$ yang digunakan untuk memodifikasi himpunan V yang berisi keseluruhan koefisien DCT dalam satu frame, dari koefisien ke $(k+1)$ sampai dengan koefisien ke $(k+n)$ berdasarkan persamaan berikut :

$$v_{x,i} = v_i + \alpha |v_i| x_i \tag{4.1}$$

di mana v_i merupakan koefisien DCT awal, x_i sampel data masukan, $v_{x,i}$ koefisien yang telah dimodifikasi, dan α merupakan parameter energi untuk *watermark*, semakin tinggi α maka *watermark* tersebut akan semakin kuat dan tampak pada citra. Persamaan 4.1 dapat dituliskan dalam bentuk perkalian vektor X dan V : $V_x = V + \alpha X|V|$.

Simulasi dilakukan dengan mentransformasikan data satu dimensi kemudian dilanjutkan dengan *threshold coding*. *Watermark* berupa urutan data bilangan real $X = \{x_1, x_2, \ldots, x_n\}$, di mana tiap nilai x_i diperoleh dari proses enkripsi yang diskalakan menjadi bilangan real antara 0 sampai dengan 1. Proses penyisipan *watermark* dapat dipandang seperti proses komunikasi yang terdiri dari dua langkah penting, langkah pertama adalah *watermark casting* di mana sinyal, yaitu data *watermark*, ditransmisikan melalui channel, dalam hal ini citra itu sendiri. Yang kedua adalah *watermark detection* yakni prosedur pendeteksian keberadaan data *watermark* di dalam citra.

---

[6] J. Zhao and E. Koch, Proceeding of the Int. Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Technology, "Embedding robust labels into images for copyright protection", Vienna, August 1995, 242-252.

4.1 Watermark Casting

Di dalam *watermark casting* dilakukan pencarian matrik DCT untuk citra *gray scale* I berukuran $N \times N$. Koefisien DCT tersebut kemudian disusun kembali dalam urutan zig-zag, mirip dengan algoritma kompresi JPEG. Kemudian diambil koefisien ke $(k+1)$ sampai dengan yang ke $(k+n)$ untuk mendapatkan vektor $T = \{t_{k+1}, \ldots, t_{k+n}\}$ dan enkripsi *plaintext* yang hasilnya diskalakan menjadi bilangan real antara 0 sampai dengan 1 dilakukan untuk mendapatkan data *watermark* $X = \{x_1, x_2, \ldots, x_n\}$. Kemudian data *watermark* $X = \{x_1, x_2, \ldots, x_n\}$ disisipkan setelah k koefisien pertama (koefisien n digantikan oleh X) sehingga didapatkan vektor $T' = \{t'_{k+1}, \ldots, t'_{k+n}\}$ menurut persamaan :

$$t'_{k+i} = t_{k+i} + \alpha |t_{k+i}| x_i \tag{4.2}$$

di mana $i = 1, \ldots, n$. DCT yang termodifikasi kemudian disusun kembali dalam urutan zig-zag dan kemudian dilakukan proses invers DCT untuk memperoleh citra *watermark* I'.

4.2 Watermark Detection

Di dalam *watermark detection*, citra *watermark* I' ditransformasi dengan matrik DCT $N \times N$, kemudian koefisien yang diperoleh diatur dalam urutan zig-zag. Cara ini dilakukan pada keseluruhan citra menurut aliran diagonal. Mula-mula dipilih elemen pada posisi sudut kiri atas atau posisi (0,0), yang merupakan nilai tertinggi, dilanjutkan sampai pada posisi sudut kanan bawah sebagai nilai yang terendah. Kemudian serangkaian koefisien pertama k dilewati

dan koefisien ke $(k+1)$ sampai dengan yang ke $(k+n)$ diambil untuk mendapatkan vektor $T^* = \{t^*_{k+1}, t^*_{k+2}, \ldots, t^*_{k+n}\}$. Korelasi $\rho(X, T^*)$ antara *watermark* X dan koefisien-koefisien DCT yang ditandai T^*, didefinisikan sebagai berikut :

$$\rho(X, T^*) = \frac{X \cdot T^*}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i t_i^* \qquad (4.3)$$

Kemudian dilakukan penghitungan terhadap korelasi $\rho(X, T^*)$ dan dibandingkan dengan nilai *threshold* T_ρ. Untuk mendeteksi keberadaan data *watermark* di dalam citra, dilakukan yang berikut :

apabila nilai korelasi $\rho(X, T^*)$ lebih kecil daripada nilai *threshold* T_ρ, maka dekoder memutuskan bahwa citra tidak ditandai dengan *watermark* X. Sebaliknya jika nilai korelasi $\rho(X, T^*)$ lebih tinggi daripada nilai *threshold* T_ρ, maka dekoder mengasumsikan bahwa citra ditandai dengan *watermark* X.

BAB V

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

BAB V

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

Perangkat lunak yang dikembangkan dalam tugas akhir ini adalah untuk mengimplementasikan sistem proteksi data citra dengan menggunakan metode *Watermarking*. Sistem proteksi data citra ini meliputi penyisipan data *watermark* ke dalam citra dan pendeteksian keberadaan data *watermark* di dalam citra, sebagaimana yang telah diuraikan dalam bab-bab sebelumnya. Perangkat lunak tersebut dibangun dengan menggunakan bahasa pemrograman Borland C++ Builder versi 3.0 dalam sistem operasi Windows NT versi 4.0

5.1 Tujuan dan Sasaran Sistem

Perangkat lunak yang akan dibuat dirancang untuk dapat memproteksi data citra pada PC (*Personal Computer*) dan secara efektif mendeteksi keberadaan data *watermark* yang disisipkan ke dalam citra, dengan metode *Watermarking*. Metode ini masih tergolong baru dalam upaya memproteksi data citra, yang meliputi dua proses penting yaitu *watermark casting* dan *watermark detection*.

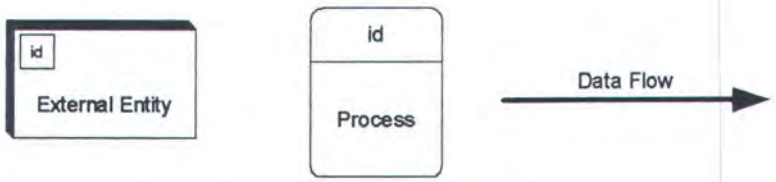
5.2 Kebutuhan Sistem

Sistem perangkat keras yang dibutuhkan untuk menjalankan perangkat lunak ini dengan baik adalah sebuah IBM PC atau kompatibelnya, dengan spesifikasi *microprocessor* minimal generasi *pentium*. Spesifikasi yang lebih rendah sebenarnya bisa dipakai, tetapi akan mengalami kelambatan yang menjemukan.

Sedangkan untuk kecepatan proses, selain *microprocessor* yang diajukan di atas, diperlukan kapasitas memori yang cukup. Manajemen memori yang dilakukan oleh *Windows* sangat bagus sehingga masalah ini dapat sedikit tertangani. Karena selain menggunakan memori dalam RAM yang ada, *Windows* dapat memanfaatkan ruang kosong *harddisk* sebagai *virtual memory* (memori semu). Dapat diusulkan di sini, untuk tipe *microprocessor* yang diajukan di atas, sebaiknya kapasitas memori RAM yang digunakan minimal 8 *Megabytes*.

5.3 Deskripsi Sistem

Sub bab ini mendeskripsikan sistem perangkat lunak dengan menggunakan Diagram Aliran Data (*Data Flow Diagram*-DFD). Simbol-simbol yang digunakan mengacu pada *Gane-Sarson* DFD, yang terdiri atas



Gambar 5.1 Komponen *Gane-Sarson* DFD

Entitas (kesatuan luar) atau *boundary* (batas sistem), memisahkan sistem dengan lingkungan luarnya. Sistem akan menerima masukan dan menghasilkan keluaran kepada lingkungan luarnya.

Process (proses), adalah kegiatan atau kerja yang dilakukan oleh orang, mesin atau komputer terhadap suatu arus data yang masuk ke dalam proses untuk menghasilkan arus data terproses (olahan) yang akan keluar dari proses.

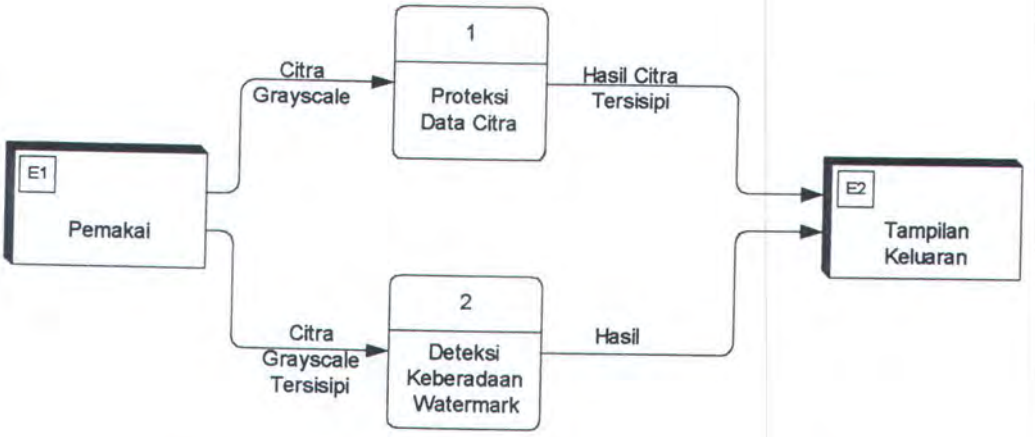
Data Flow (aliran data), menunjukkan aliran dari data yang dapat berupa masukan untuk sistem atau hasil dari proses sistem. Aliran data dapat terjadi diantara proses, simpanan data dan kesatuan luar. Tanda panah menunjukkan arah aliran data.

DFD level 0 dari aplikasi proteksi data citra dengan metode *Watermarking* dideskripsikan pada Gambar 5.2. Sistem proteksi data menerima masukan dari pemakai dalam bentuk citra (citra *gray scale*, yang belum dimodifikasi agar sesuai kebutuhan sistem). Selanjutnya citra tersebut dimodifikasi oleh sistem, hasilnya kemudian ditampilkan.

Gambar 5.2 DFD level 0, aplikasi proteksi data citra

Proses proteksi dideskripsikan secara lebih rinci pada Gambar 5.3 dalam DFD level 1. Pada proteksi data terdapat dua proses, yaitu proteksi data citra dan

deteksi keberadaan *watermark*. Proses proteksi data citra merupakan proses memodifikasi citra dengan cara menyisipkan data *watermark* ke dalam citra. Proses proteksi data citra ini menghasilkan citra tersisipi, dan ditampilkan.

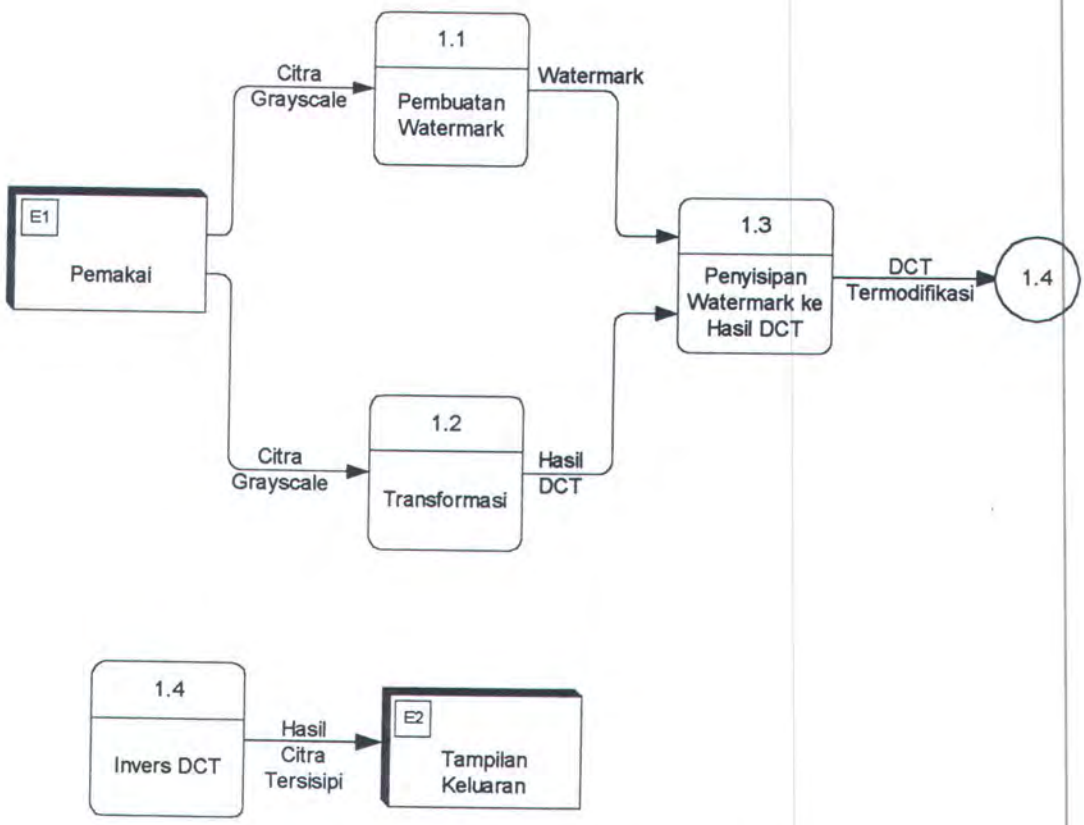


Gambar 5.3 DFD level 1, aplikasi proteksi data citra

Proses deteksi keberadaan *watermark* merupakan proses untuk mendeteksi keberadaan data *watermark* yang disisipkan ke dalam citra. Hasil dari proses pendeteksian keberadaan *watermark* ini kemudian ditampilkan dalam bentuk grafik.

Selanjutnya proses proteksi data citra pada Gambar 5.3 dideskripsikan secara lebih rinci pada Gambar 5.4, dalam DFD level 2. Proses proteksi data citra terdiri atas empat proses, yaitu pembuatan *watermark*, transformasi, penyisipan *watermark* ke hasil DCT, dan invers DCT. Proses pembuatan *watermark* merupakan proses enkripsi *plaintext* dengan menggunakan kunci *private* dan menghasilkan urutan tanda ASCII yang kemudian diskalakan ke dalam bilangan real antara 0 sampai dengan 1. Caranya adalah dengan membagi tiap-tiap tanda

ASCII dengan nilai 255 yang kemudian disisipkan ke dalam citra sebagai data *watermark*.

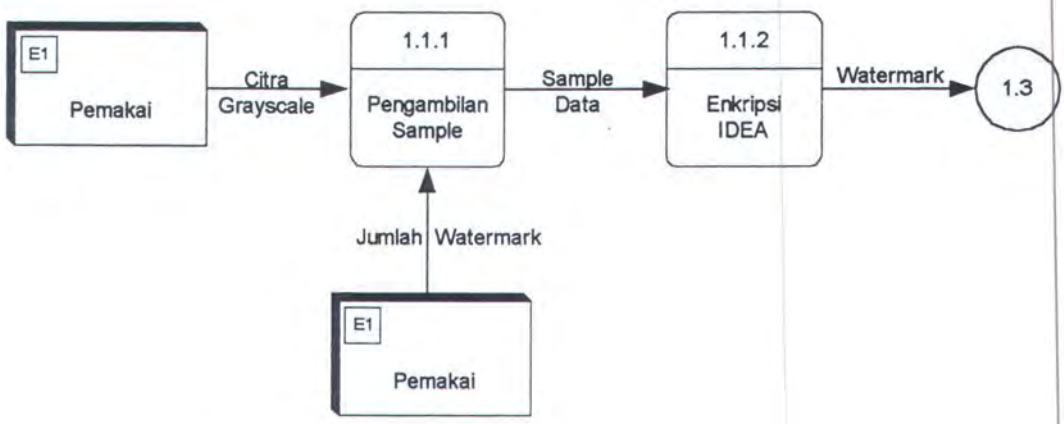


Gambar 5.4 DFD level 2, proteksi data citra

Sedangkan transformasi mengolah titik-titik dari *domain spasial* dan mentransformasikannya ke bentuk sejenis dalam *domain frekuensi*. *Watermark* dan DCT yang dihasilkan digunakan sebagai masukan dalam proses penyisipan *watermark* ke hasil DCT. Proses penyisipan *watermark* ke hasil DCT merupakan proses memodifikasi beberapa koefisien DCT yang terpilih dengan data *watermark*

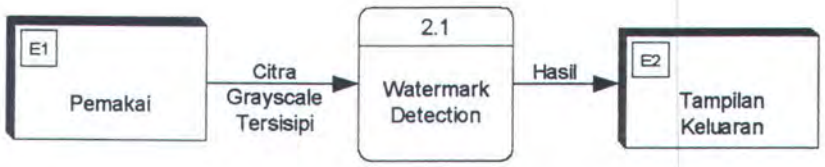
menurut aturan *Watermarking* dan menghasilkan DCT termodifikasi. Kemudian dilakukan invers DCT untuk menghasilkan citra tersisipi, dan ditampilkan.

Proses pembuatan *watermark* pada Gambar 5.4 secara lebih detail dideskripsikan pada Gambar 5.5, dalam DFD level 3. Proses pembuatan *watermark* ini didahului dengan pemakai memasukkan jumlah *watermark*, yaitu pada proses pengambilan sample. Proses pengambilan sample ini menghasilkan sample data sebanyak jumlah *watermark*, yang digunakan sebagai masukan dalam proses enkripsi IDEA.



Gambar 5.5 DFD level 3, pembuatan watermark

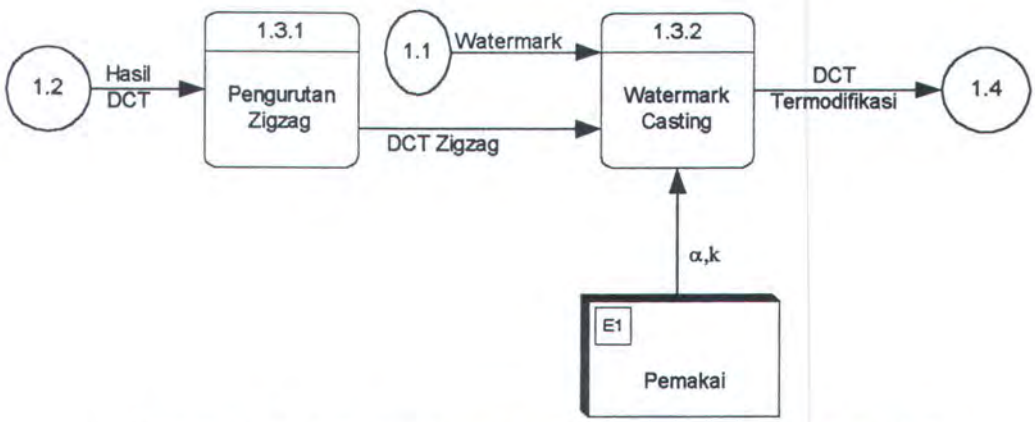
Proses enkripsi IDEA mengenkripsi secara berturut-turut block-block 64 bit dari sample data dengan menggunakan sebuah *key* yang panjangnya 128-bit, dan menghasilkan urutan tanda ASCII yang kemudian diskalakan ke dalam bilangan real antara 0 sampai dengan 1 yang digunakan sebagai data *watermark*.



Gambar 5.6 DFD level 2, deteksi keberadaan watermark

Proses deteksi keberadaan *watermark* dideskripsikan secara lebih rinci pada Gambar 5.6 dalam DFD level 2. Deteksi keberadaan *watermark* melewati satu proses saja, yaitu *watermark detection*. Kemudian hasil pendeteksian ditampilkan dalam bentuk grafik.

Proses penyisipan *watermark* ke hasil DCT pada Gambar 5.4 secara lebih rinci dideskripsikan pada Gambar 5.7, dalam DFD level 3. Proses ini terdiri atas proses pengurutan zigzag dan proses *watermark casting*. Proses pengurutan zigzag merupakan proses mengurutkan koefisien-koefisien matrik DCT secara zigzag.



Gambar 5.7 DFD level 3, penyisipan watermark ke hasil DCT

Pemakai memberikan nilai energi *watermark* (α) dan banyaknya koefisien yang dilewati (k) serta data *watermark* dan DCT zigzag sebagai masukan dalam

proses *watermark casting*. Proses *watermark casting* memodifikasi koefisien DCT ke (k+1) sampai dengan koefisien ke (k+n) menurut persamaan 4.2 dan menghasilkan DCT termodifikasi.

5.4 Perancangan Perangkat Lunak

Secara garis besar, metodologi perancangan perangkat lunak ini dapat dimulai dengan pendefinisian DCT, pemanggilan file input, penyimpanan file ke array satu dimensi. Array titik dari file input dikalikan dengan matrik DCT untuk menghasilkan frekuensi. Frekuensi ini disusun dalam urutan zig-zag, dihasilkan data *watermark*, dimodifikasi dan akhirnya dihasilkan citra yang tersisipi data *watermark*.

5.4.1 Implementasi DCT

Salah satu sifat transformasi citra yang akan dimanfaatkan di sini adalah pemampatan energi pada frekuensi rendah. Konsentrasi sajian citra terletak pada koefisien kiri atas dari matrik *output*, sehingga tidak mungkin implementasi DCT dilakukan dengan membagi citra menjadi bagian yang lebih kecil atau block-block. Karena penerapan DCT pada block-block atau bagian yang lebih kecil dari citra, menyebabkan konsentrasi sajian terletak pada koefisien kiri atas dari masing-masing block, bukan pada koefisien kiri atas dari citra. Akhirnya penerapan DCT dilakukan pada keseluruhan citra sekaligus.

5.4.2 Perkalian Matrik

Definisi DCT yang diberikan pada Gambar 3.3 kelihatannya terlalu kaku, dengan perulangan ganda. Elemen di dalam perulangan dapat dikerjakan $N \times N$ kali untuk setiap elemen DCT yang akan dihitung. Baris tengah dari perulangan memiliki dua operasi perkalian dan sebuah operasi penjumlahan.

Terdapat bentuk DCT yang lebih sederhana untuk mengerjakan operasi matrik tersebut. Untuk melakukan operasi ini, pertama-tama dibuat matrik $N \times N$ yang disebut dengan *CosMatrix*, Matrik Transformasi Cosinus, C.

Matrik ini didefinisikan dengan persamaan di bawah ini.

$$C_{i,j} = \begin{bmatrix} \frac{1}{\sqrt{N}} & \text{untuk } i = 0 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{(2j+1)i\pi}{2N}\right] & \text{untuk } i > 0 \end{bmatrix}$$

Setiap Matrik Transformasi Cosinus dibuat, sekaligus ditranpose dengan memutar matrik tersebut terhadap diagonal, dengan kata lain, baris matrik tersebut menjadi kolom. Matrik ini kita sebut *CosTrMatrix*, Matrik Transpose Transformasi Cosinus. Pembuatan matrik ini dikerjakan sekali selama inisialisasi program. Kedua matrik ini dapat dibuat pada waktu yang bersamaan dengan perulangan yang cukup pendek. Algoritmanya yang digunakan adalah sebagai berikut :

Misal jumlah baris dalam matrik N

1. Untuk semua kolom j dalam matrik kerjakan langkah 2
2. $CosTrMatrix_{j,0} = CosMatrix_{0,j} = \frac{1}{\sqrt{N}}$
3. Untuk semua elemen i, j dalam matrik di mana $i > 0$ kerjakan langkah 4

4. $CosTrMatrix_{j,i} = CosMatrix_{i,j} = \sqrt{\frac{2}{N}} Cos\left[\frac{(2j+1)i\pi}{2N}\right]$

5. selesai

Setelah matrik ini selesai dibuat, definisi fungsi DCT menjadi :

DCT = *CosTrMatrix* × *Piksel* × *CosMatrix*

Untuk persamaan ini, operator '×' , berarti perkalian matrik, bukan perkalian aritmatik biasa. Setiap faktor dalam persamaan ini adalah matrik $N \times N$.

Pada saat mengalikan dua matrik bujursangkar, nilai operasi aritmatik masing-masing elemen dari matrik *output* akan menjadi N operasi perkalian dan N operasi penjumlahan. Karena melakukan dua perkalian matrik untuk membuat matrik DCT, maka masing-masing elemen dari matrik transformasi DCT diperoleh dengan $2N$ perkalian dan $2N$ penjumlahan.

1. Untuk semua nilai i dalam matrik kerjakan langkah 2
2. Untuk semua nilai j dalam matrik kerjakan langkah 3 dan 4
3. $temp_{i,j} = 0.0$
4. kerjakan langkah 5 untuk $k = 0$ sampai $k = n-1$
5. $temp_{i,j} = temp_{i,j} + piksel_{i,k} \times CosTrMatrix_{k,j}$
6. Untuk semua nilai i dalam matrik kerjakan langkah 7
7. Untuk semua nilai j dalam matrik kerjakan langkah 8 dan 9
8. $temp_{i,j} = 0.0$
9. Kerjakan langkah 10 untuk $k = 0$ sampai $k = n-1$
10. $temp_{i,j} = temp_{i,j} + CosMatrix_{k,j} \times piksel_{i,k}$

11. $DCT_{i,j} = temp_{i,j}$

12. Selesai

Algoritma yang mengimplementasikan DCT melalui aritmatika matrik, diperlihatkan di atas. Perhatikan bahwa algoritma tersebut di atas sebenarnya tidak lebih dari kumpulan dari dua block perulangan, di mana tiap block terdiri dari tiga perulangan. Block pengulangan pertama mengalikan kumpulan matrik input dengan Matrik Tranpose Transformasi Cosinus, membuat matrik temporary kemudian block pengulangan kedua mengalikan Matrik Transformasi Cosinus dengan matrik temporary, yang menghasilkan *output* matrik DCT.

5.4.3 Output Dari DCT

Contoh matrik input berupa *Matrik Piksel* berikut ini menyajikan contoh sebuah block input representatif dari sebuah citra. Sebagaimana terlihat, contoh matrik 8×8 piksel yang diacak berkisar 140 hingga 175. Nilai-nilai integer ini akan diolah dengan algoritma DCT, membuat *Matrik output* seperti yang ditunjukkan pada Gambar 5.8.

Koefisien yang terletak pada posisi (0,0) yaitu pada posisi sudut kiri atas dari matrik dinamakan koefisien DC. Nilai ini menunjukkan rata-rata dari keseluruhan magnitude dari matrik input. Perhatikan bahwa Koefisien DC berkecenderungan lebih besar dari nilai-nilai yang lain dari dalam matrik DCT. Terdapat *trend* yang umum dalam matrik DCT. Seiring dengan berjalannya elemen dari koefisien DC, nilai-nilai magnitude ini cenderung menjadi lebih kecil.

Contoh Matrik Input berupa Matrik Pixel :

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

Contoh Matrik Output DCT :

186	-18	14.78	-8.98	23.25	-9.23	-13.97	-18.94
20.54	145	26.33	-9.04	-10.93	10.73	13.77	6.95
-10.38	-23.51	-1.85	6.04	-18.07	3.2	-20.42	-0.83
-8.1	-5.04	14.33	-14.61	-8.22	-2.73	-3.08	8.43
-3.25	9.5	7.88	1.32	-11	17.9	18.38	15.24
3.86	-2.21	-18.17	8.5	8.27	-3.61	0.87	-6.86
8.9	0.63	-2.92	3.64	-1.17	-7.42	-1.15	-1.92
0.05	-7.81	-2.42	1.59	1.2	4.25	-6.42	0.31

Gambar 5.8 Contoh DCT pada suatu blok gambar

Walaupun kelihatannya kompleks, implementasi DCT sebenarnya sederhana. Pertama-tama dilakukan perkalian data matrik input, berupa matrik piksel, dengan Matrik Transpose Transformasi Cosinus dan hasil perkalian tersebut disimpan pada matrik temporary $N \times N$. Selanjutnya, matrik temporary dikalikan dengan Matrik Transformasi Cosinus dan hasilnya disimpan pada *Matrik Output*.

141	144	143	139	146	161	172	173
149	145	143	145	150	157	166	173
159	146	143	154	156	150	156	172
167	146	144	160	160	144	147	168
166	147	145	161	160	142	143	161
158	147	146	156	155	143	142	153
147	146	147	149	149	146	145	146
139	146	148	144	145	149	147	141

Gambar 5.9 Nilai pixel sesudah IDCT

Invers DCT dilakukan dengan proses kebalikan dari DCT. Pertama kita kalikan nilai-nilai DCT dengan $N \times N$ Matrik Transformasi Cosinus, hasil perkalian ini dikalikan lagi dengan Matrik Transpose Transformasi Cosinus.

Ini berarti dengan melakukan DCT pada data input, konsentrasi sajian gambar terletak pada koefisien kiri atas dari *Matrik Output*, dengan koefisien yang terletak di sebelah kanan bawah dari nilai tersebut mengandung informasi yang tidak seberapa penting.

5.5 Pembuatan Watermark

Rangkaian bilangan random antara 0 sampai dengan 255 merupakan data yang akan dienkripsi dan disebut *plaintext*. Tiap block *plaintext* yang panjangnya 64-bit digunakan sebagai input untuk algoritma IDEA.

Block *plaintext* 64-bit dibagi ke dalam empat sub-block 16-bit, operasi aljabar XOR, penjumlahan modulo 2^{16} dan perkalian modulo $2^{16}+1$ digunakan di dalam proses enkripsi pada angka-angka 16-bit. Setiap putaran enkripsi menghasilkan enam *subkey* 16-bit dari *key* 128-bit. Empat *subkey* 16-bit berikutnya merupakan *output* transformasi, ada 52 (= 8 * 6 + 4) *subkey* 16-bit yang berbeda dihasilkan dari *key* 128-bit. Putaran enkripsi pertama, empat *subkey* 16-bit pertama bergabung dengan dua block *plaintext* 16-bit menggunakan penjumlahan modulo 2^{16}, dan dengan dua block *plaintext* 16-bit yang lain menggunakan perkalian modulo $2^{16}+1$. Dua *subkey* 16-bit sisanya masuk dalam perhitungan dan operator aljabar grup yang ketiga, digunakan untuk XOR bit demi bit. Di akhir putaran pertama enkripsi dihasilkan empat nilai 16-bit yang digunakan sebagai input untuk putaran kedua enkripsi. Proses yang digambarkan di atas untuk putaran satu diulang di dalam setiap putaran berikutnya dari 7 putaran enkripsi yang menggunakan *subkey* 16-bit yang berbeda untuk setiap kombinasi. Pada waktu *output* transformasi yang berikut, empat nilai 16-bit yang dihasilkan di akhir putaran enkripsi ke delapan bergabung dengan empat yang terakhir dari 52 *subkey* menggunakan penjumlahan modulo 2^{16} dan perkalian modulo $2^{16}+1$ ke bentuk

yang menghasilkan empat sub-block *ciphertext* 16-bit. Sub-block 16-bit yang semuanya berisi bit 0, diartikan sebagai 2^{16}.

Ada 52 *subkey* 16-bit dihasilkan dari *key* 128-bit. Pertama, *key* 128-bit dibagi ke dalam delapan *subkey* 16-bit, kemudian langsung digunakan sebagai delapan *subkey* pertama. Kemudian terhadap *key* 128-bit dilakukan rotasi ke kiri sebanyak 25 bit, dan hasil rotasi sebesar 128 bit itu dibagi lagi ke dalam delapan *subkey* 16-bit yang langsung digunakan sebagai delapan *subkey* berikutnya. Prosedur rotasi 25-bit ke kiri diulang sampai dihasilkan sebanyak 52 *subkey* 16-bit.

Proses komputasi yang digunakan untuk dekripsi dari *ciphertext* sama seperti untuk enkripsi dari *plaintext*. Yang berbeda hanya *subkey* 16-bit yang digunakan.

5.5.1 Gambaran dari IDEA

Block data 64-bit dibagi ke dalam empat sub-block 16-bit yaitu X_1, X_2, X_3 dan X_4. Empat sub-block ini menjadi input untuk putaran pertama dari algoritma. Total ada delapan putaran.

Di dalam setiap putaran, empat sub-block di XOR, ditambah, dan dikalikan dengan yang lain dan dengan enam *subkey* 16-bit. Diantara setiap putaran, sub-block kedua dan ketiga di swap.

Di setiap putaran, urutan kejadian adalah sebagai berikut :

1. Mengalikan X_1 dan *subkey* pertama.
2. Menjumlahkan X_2 dan *subkey* kedua.
3. Menjumlahkan X_3 dan *subkey* ketiga.

4. Mengalikan X_4 dan *subkey* keempat.
5. XOR hasil dari step 1 dan 3.
6. XOR hasil dari step 2 dan 4.
7. Mengalikan hasil dari step 5 dengan *subkey* kelima.
8. Menjumlahkan hasil dari step 6 dan 7.
9. Mengalikan hasil dari step 8 dengan *subkey* keenam.
10. Menjumlahkan hasil dari step 7 dan 9.
11. XOR hasil dari step 1 dan 9.
12. XOR hasil dari step 3 dan 9.
13. XOR hasil dari step 2 dan 10.
14. XOR hasil dari step 4 dan 10.

Output dari putaran adalah empat sub-block yang merupakan hasil dari langkah 11, 13, 12, dan 14. Swap dua sub-block sebelah dalam (kecuali untuk putaran terakhir), dan merupakan input untuk putaran berikutnya.

Setelah putaran kedelapan, ada sebuah *output* terakhir transformasi :

1. Mengalikan X_1 dan *subkey* pertama.
2. Menjumlahkan X_2 dan *subkey* kedua.
3. Menjumlahkan X_3 dan *subkey* ketiga.
4. Mengalikan X_4 dan *subkey* keempat.

Yang terakhir, empat sub-block dirangkaikan untuk menghasilkan *ciphertext*. Kemudian setiap satu *byte* data *ciphertext* dibagi 255, untuk menghasilkan bilangan real yang merupakan data *watermark*.

5.6 Penyisipan Watermark ke Hasil DCT

Pada tahapan ini proses pertama yang dilakukan adalah mengurutkan koefisien-koefisien matrik DCT secara zigzag. Kemudian hasil DCT yang terurut secara zigzag bersama-sama dengan data *watermark*, energi *watermark* (α) dan banyaknya koefisien yang dilewati (k) digunakan sebagai masukan untuk proses selanjutnya yaitu *watermark casting*. Di dalam *watermark casting* dilakukan modifikasi terhadap matrik DCT yang tersusun secara zigzag. Caranya adalah dengan menyisipkan data *watermark* ke dalam matrik DCT pada koefisien ke $(k+1)$ sampai dengan yang ke $(k+n)$ untuk mendapatkan vektor $T^{!} = \{t^{!}_{k+1}, t^{!}_{k+2}, t^{!}_{k+3} \ldots, t^{!}_{k+n}\}$ menurut persamaan :

$$t^{'}_{k+i} = t_{k+i} + \alpha |t_{k+i}| x_i \qquad (5.1)$$

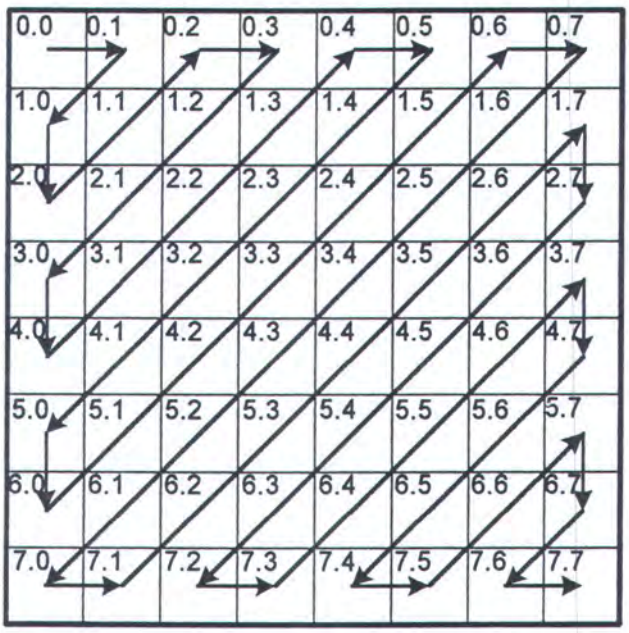
di mana $i = 1,\ldots,n$. DCT yang termodifikasi kemudian disusun kembali dalam urutan zig-zag dan kemudian dilakukan proses invers DCT untuk memperoleh citra *watermark* $I^{'}$.

5.6.1 Urut-Urutan ZIGZAG

Satu alasan mengapa cara ini digunakan untuk menentukan koefisien-koefisien DCT yang akan dimodifikasi dengan data *watermark* adalah karena sifat dari hasil transformasi. Konsentrasi sajian citra terletak pada koefisien kiri atas dari matrik *output*, sedang koefisien yang terletak di sebelah kanan bawah dari nilai tersebut mengandung informasi yang tidak seberapa penting. Semakin jauh dari posisi (0,0) , semakin sedikit kontribusi elemen terhadap citra. Sehingga

penyisipan data *watermark* dapat dilakukan pada koefisien-koefisien yang sedikit memberi kontribusi terhadap citra.

Jalur sebenarnya dari urutan zig-zag ini seperti contoh pada Gambar 5.10 berikut ini.



Gambar 5.10 Urut-urutan ZigZag

Pada algoritma tugas akhir ini, urutan akan dimodifikasi dengan sebagian struktur yang dapat diakses secara sekuensial yang baris dan kolomnya dibaca dengan cara sebagai berikut :

```
Zig_Zag[ 8 * 8 ] =
{
    [0 , 0] ,
    [0 , 1] , [1 , 0] ,
    [2 , 0] , [1 , 1] , [0 , 2] ,
    [0 , 3] , [1 , 2] , [2 , 1] , [3 , 0] ,
    [4 , 0] , [3 , 1] , [2 , 2] , [1 , 3] , [0 , 4] ,
    [0 , 5] , [1 , 4] , [2 , 3] , [3 , 2] , [4 , 1] , [5 , 0] ,
    [6 , 0] , [5 , 1] , [4 , 2] , [3 , 3] , [2 , 4] , [1 , 5] , [0 , 6] ,
    [0 , 7] , [1 , 6] , [2 , 5] , [3 , 4] , [4 , 3] , [5 , 2] , [6 , 1] , [7 , 0] ,
    [7 , 1] , [6 , 2] , [5 , 3] , [4 , 4] , [3 , 5] , [2 , 6] , [7 , 1] ,
    [2 , 7] , [3 , 6] , [4 , 5] , [5 , 4] , [6 , 3] , [7 , 2] ,
    [7 , 3] , [6 , 4] , [5 , 5] , [4 , 6] , [3 , 7] ,
    [4 , 7] , [5 , 6] , [6 , 5] , [7 , 4] ,
    [7 , 5] , [6 , 6] , [5 , 7] ,
    [6 , 7] , [7 , 6] ,
    [7 , 7] ,
}
```

Tiap-tiap hasil modifikasi, akan ditentukan oleh baris dan kolom yang digunakan berikutnya dengan memperhatikan struktur zig-zag. Selanjutnya akan dimodifikasi elemen yang ditentukan dengan baris dan kolom dari struktur zig-zag.

5.7 Pendeteksian Keberadaan Data Watermark

Dalam proses deteksi keberadaan *watermark* langkah pertama yang dilakukan adalah menggunakan citra *grayscale* yang tersisipi data *watermark* sebagai masukan untuk proses *watermark detection*. Di dalam *watermark detection* ini citra *grayscale* yang tersisipi ditransformasikan dengan matrik DCT $N \times N$, kemudian koefisien yang diperoleh diatur dalam urutan zig-zag. Serangkaian koefisien pertama k dilewati, dan koefisien ke $(k+1)$ sampai dengan yang ke $(k+n)$ diambil untuk mendapatkan vektor $T^* = \{t^*_{k+1}, t^*_{k+2}, \ldots, t^*_{k+n}\}$. Korelasi

$\rho(X,T^*)$ antara *watermark* X dan koefisien-koefisien DCT yang ditandai T^*, didefinisikan sebagai berikut :

$$\rho(X,T^*)=\frac{X\cdot T^*}{n}=\frac{1}{n}\sum_{i=1}^{n}x_i t_i^* \tag{5.2}$$

Kemudian dilakukan perhitungan terhadap korelasi $\rho(X,T^*)$ dan dibandingkan dengan nilai *threshold* T_ρ. Untuk mendeteksi keberadaan data *watermark* di dalam citra, dilakukan yang berikut :

apabila nilai korelasi $\rho(X,T^*)$ lebih kecil daripada nilai *threshold* T_ρ, maka dekoder memutuskan bahwa citra tidak ditandai dengan *watermark* X. Sebaliknya jika nilai korelasi $\rho(X,T^*)$ lebih tinggi dari *threshold* T_ρ, maka dekoder mengasumsikan bahwa citra ditandai dengan *watermark* X.

5.8 Pembuatan Perangkat Lunak

Perangkat lunak yang dibuat dalam tugas akhir ini digunakan untuk menghasilkan citra *Bitmap* (BMP) yang tersisipi oleh data *watermark* dan dapat mendeteksi keberadaan data *watermark* di dalam citra. Hal ini sesuai dengan apa yang telah dicantumkan pada batasan masalah. Dengan menginputkan file citra *Bitmap* (BMP) yang akan dimodifikasi disertai dengan memasukkan nilai energi *watermark* (α), jumlah data *watermark* dan banyaknya koefisien yang dilewati (k) pada dialog input yang telah disediakan, maka akan didapatkan citra *Bitmap* (BMP) yang tersisipi oleh data *watermark.* Sedangkan deteksi keberadaan data *watermark* yang disisipkan ke dalam citra dapat dilihat dari grafik *Detector*

Response Of The Watermarked Image yang ditampilkan di layar. Dari sana dapat dilihat pengaruh besar energi *watermark* dan banyaknya data *watermark* yang disisipkan serta banyaknya koefisien yang dilewati terhadap penentuan keberadaan data *watermark* di dalam citra.

Deklarasi struktur data untuk *watermarking* dalam C++ Builder adalah sebagai berikut

```
#define  ROUNDS     8
#define  KEYLEN     (6*ROUNDS+4)

typedef   unsigned short int IDEAkey[KEYLEN];        // 52 words
typedef   unsigned short int uint16;
typedef   unsigned short int IntPlainTextType[4];
typedef   unsigned char      PlainTextType[9];
typedef   unsigned short int IntKeyType[8];          // 128 bits

extern IDEAkey DecSubkeys, EncSubkeys;
extern PlainTextType PixelSize, Result;
extern IntPlainTextType PlainText, CipherText;
extern IntKeyType KEY;

extern int MatSize;
extern float *CosMatrik, *CosTrMatrik;

class TMDI_Child : public TForm
{
__published:       // IDE-managed Components
    TPageControl *PageControl;
    TTabSheet *tabFileInput;
    TScrollBox *ScrollBoxInput;
    TImage *ImageInput;
    TTabSheet *tabFileWatermark;
    TScrollBox *ScrollBoxWatermark;
    TImage *ImageWatermark;
    TTabSheet *tabDeteksi;
    TChart *Chart1;
    TLineSeries *Series1;
    TLineSeries *Series3;
    TProgressBar *Sruut;
    void __fastcall FormClose(Tobject *Sender, TCloseAction
     &Action);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);

private:    // User declarations
    int Steps;
    float *ArrayDct;
    void __fastcall EncryptWatermark(unsigned char *Tanda_Air);
    void __fastcall DecryptWatermark(unsigned char *Tanda_Air);
    void __fastcall Application_Detector(float Th_Baru,float *z);
```

```
    void __fastcall de_alokasi_Piksel(unsigned char
    **DataPixel,int Baris);
    void __fastcall Threshold_Baru(unsigned int WaterSize,float
    *KoefisienModify, float *Th_Baru);
    void __fastcall Modify_DCT(float *SigSag,unsigned char
    *Tanda_Air);
    void __fastcall Modify_Ulang(float *SigSag,float
    *DCTawal,unsigned char *Tanda_Air);
    float __fastcall GetPSNR();

public:          // User declarations
    float Alpha, SruutPos, SruutRel;
    int WatermarkSize, KoefisienSize;
    __fastcall TMDI_Child(TComponent* Owner,String Name,AnsiString
Gambar);
    void __fastcall SetBlockPixel();
    void __fastcall Tanpa_Sisipan();
    void __fastcall NextSruut();
};
```

Adapun *routine-routine* utama yang digunakan di dalam perangkat lunak ini adalah sebagai berikut :

5.8.1 Inisialisasi

Yang pertama kali dikerjakan oleh perangkat lunak ini adalah mengisi *CosMatrik* dan *CosTrMatrik*. Matrik-matrik ini digunakan oleh *routine-routine CosTransform* dan *InvCosTransform*. Pengisian Matrik ini sesuai dengan rumus yang telah diterangkan pada halaman 41.

```
void __fastcall Initialisasi(int lebar,int tinggi)
{
    if(lebar >= tinggi) MatSize = lebar;
    else MatSize = tinggi;
    CosMatrik = new float[MatSize*MatSize];
    if(!CosMatrik)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1);  /* terminate program if out of memory */
    }
    CosTrMatrik = new float[MatSize*MatSize];
    if(!CosTrMatrik)
```

```
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1);  /* terminate program if out of memory */
    }
}

void __fastcall InitMatrik()
{
   int i, j;

   for(i = 0; i < MatSize; i++)
    *(CosMatrik+i) = *(CosTrMatrik+i*MatSize) = 1.0/sqrt(MatSize);

   for(i = 1; i < MatSize; i++)
      for(j = 0; j < MatSize; j++)
      {
         *(CosMatrik+(i*MatSize+j)) = *(CosTrMatrik+(i+MatSize*j))
         = sqrt(2.0/MatSize)* cos((2*j+1)*i*M_PI/(2.0*MatSize));
      }
}
```

5.8.2 Transformasi Cosinus

Walaupun kelihatannya kompleks, implementasi DCT sebenarnya sederhana. Pertama-tama data matrik input, berupa matrik piksel, dikalikan dengan *CosTrMatrik* dan hasil perkalian tersebut disimpan pada matrik temporary $N \times N$. Selanjutnya, matrik temporary dikalikan dengan *CosMatrik* dan hasilnya disimpan di *Matrik Output*.

```
void __fastcall CosTransform(unsigned char *src, float *dst)
{
    float *Buffer;
    Buffer = new float[MatSize*MatSize];
    if(!Buffer)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1);  /* terminate program if out of memory */
    }
    float *wdst = Buffer, *wdst2, *wcos, *wf, sum, bit;
    unsigned char *wbit;
    int p, q, r;

    for(p = 0; p < MatSize; wdst++, p++)
    {
        wbit = src;
        wdst2 = wdst;

        for(q = 0; q < MatSize; wdst2 += MatSize, q++)
        {
            wcos = CosTrMatrik + p;
            sum = 0.0;

            for(r = 0; r < MatSize; wcos += MatSize, r++)
            {
                bit = (unsigned char)(*(wbit++));
                sum += (*wcos * bit);
            }
            *wdst2 = sum;
        }
    }
    wdst = dst;
    for(p = 0; p < MatSize; wdst++, p++)
    {
        wcos = CosMatrik;
        wdst2 = wdst;

        for(q = 0; q < MatSize; wdst2 += MatSize, q++)
        {
            wf = Buffer + p;
            sum = 0.0;
```

```
            for(r = 0; r < MatSize; wf += MatSize, r++)
               sum += (*wf * *(wcos++));
            *wdst2 = sum;
         }
      }
   }
```

5.8.3 Subkey Enkripsi IDEA

Digunakan untuk menghasilkan sebanyak 52 *subkey* 16-bit enkripsi dari *key* 128-bit. Pertama, *key* 128-bit dibagi ke dalam delapan *subkey* 16-bit yang kemudian langsung digunakan sebagai delapan *subkey* pertama. *Key* 128-bit kemudian dirotasi sebanyak 25-bit ke kiri. *Key* 128-bit yang dihasilkan dibagi lagi ke dalam delapan *subkey* 16-bit yang langsung digunakan sebagai delapan *subkey* berikutnya. Prosedur rotasi 25-bit ke kiri diulang sampai dihasilkan sebanyak 52 *subkey* 16-bit.

```
void __fastcall IDEAEncSubkeys(IntKeyType Key, IDEAkey EncSubkeys)
{
    for (int i=0; i<6; i++)
    {
        for (int j=0; j<8; j++)
            EncSubkeys[i*8+j] = Key[j];
        RotateLeft25Bit(Key);
    }
    for (int j=0; j<4; j++) EncSubkeys[48+j] = Key[j];
}
```

5.8.4 Rotasi 25 Bit ke Kiri

Digunakan untuk merotasi *key* yang panjangnya 128-bit sebanyak 25-bit ke kiri.

```
void __fastcall RotateLeft25Bit(IntKeyType Key)
{
    uint16 i, Over;
```

```
    //--- rotate 16 bit ---
    Over = Key[0];
    memcpy(Key,Key+1,7*sizeof(Key[0]));
    Key[7] = Over;

    //--- rotate 9 bit ---
    Over = (uint16)(Key[0] >> 7);
    for(i = 0; i < 7; i++)
       Key[i] = (uint16)((Key[i] << 9) | (Key[i+1] >> 7));
    Key[7] = (uint16)((Key[7] << 9) | Over);
}
```

5.8.5 Modulo Penjumlahan

Digunakan untuk menghitung penjumlahan bilangan pertama dengan bilangan kedua, kemudian dilakukan modulo dengan 2^{16}.

```
const static uint16 ModuloAddition(register uint16 A, register
uint16 B)
{
    return (uint16)((A + B) % MAX_WORD);
}
```

5.8.6 Modulo Perkalian

Digunakan untuk menghitung perkalian bilangan pertama dengan bilangan kedua, kemudian dilakukan modulo dengan $2^{16}+1$.

```
const static uint16 ModuloMultiplication(register uint16 A,
register uint16 B)
{
    register unsigned int P;
    register uint16 Result;

    if(A)
    {
        if(B)
        {
            P = A * B;
            Result = (uint16)((P % MAX_WORD_1) & 0xFFFF);
            return Result;
        }
        else return (uint16)(1-A);
    }
    else return (uint16)(1-B);
}
```

5.8.7 Invers Perkalian

Digunakan untuk menghitung invers perkalian dari suatu bilangan, modulo $2^{16}+1$.

```
const static short int inverse(uint16 x)
{
    int g[50], u[50], v[50], i=1, y;
    short int Result;

    g[0] = 0x10001; u[0] = 1; v[0] = 0;
    g[1] = x; u[1] = 0; v[1] = 1;

    if(x == 1) return x;
    while(g[i])
    {
       y = g[i-1] / g[i];
       g[i+1] = g[i-1] - (y * g[i]);
       u[i+1] = u[i-1] - (y * u[i]);
       v[i+1] = v[i-1] - (y * v[i]);
       i++;
    }
    Result = (short int)v[i-1];
    if(Result < 0) Result = (short int)(MAX_WORD_1 + Result);
    return Result;
}
```

5.8.8 Urutan Kejadian Setiap Putaran

Digunakan untuk menentukan urutan kejadian di setiap putaran. *Output* dari putaran adalah empat sub-block 16-bit. Swap dilakukan pada dua sub-block sebelah dalam (kecuali untuk putaran terakhir), dan merupakan input untuk putaran berikutnya.

```
void __fastcall RoundSteps(uint16 Round, IntPlainTextType Data,
register const IDEAkey Subkeys)
{
    register uint16 Temp[10];

    Temp[0] = ModuloMultiplication(Data[0],Subkeys[Round*6]);
    Temp[1] = ModuloAddition(Data[1],Subkeys[Round*6+1]);
    Temp[2] = ModuloAddition(Data[2],Subkeys[Round*6+2]);
    Temp[3] = ModuloMultiplication(Data[3],Subkeys[Round*6+3]);
    Temp[4] = Temp[0] ^ Temp[2];
    Temp[5] = Temp[1] ^ Temp[3];
    Temp[6] = ModuloMultiplication(Temp[4],Subkeys[Round*6+4]);
    Temp[7] = ModuloAddition(Temp[5],Temp[6]);
    Temp[8] = ModuloMultiplication(Temp[7],Subkeys[Round*6+5]);
    Temp[9] = ModuloAddition(Temp[6],Temp[8]);
    Data[0] = Temp[0] ^ Temp[8];
    Data[2] = Temp[2] ^ Temp[8];
    Data[1] = Temp[1] ^ Temp[9];
    Data[3] = Temp[3] ^ Temp[9];
}
```

5.8.9 Enkripsi atau Dekripsi IDEA

Digunakan untuk melakukan enkripsi dari *plaintext* atau dekripsi dari *ciphertext*. Tergantung *subkey-subkey* 16-bit yang digunakan, jika *subkey* enkripsi yang digunakan maka *routine* berfungsi sebagai enkripsi. Sedangkan jika *subkey* dekripsi yang digunakan maka *routine* berfungsi sebagai dekripsi.

```
void __fastcall CipherIdea(IntPlainTextType Data, IntPlainTextType
out, register const IDEAkey Subkeys)
{
    register uint16 Round;

    for (Round=0; Round<8; Round++)
        RoundSteps(Round,Data,Subkeys);

    *out++ = ModuloMultiplication(Data[0],Subkeys[48]);
    *out++ = ModuloAddition(Data[1],Subkeys[49]);
    *out++ = ModuloAddition(Data[2],Subkeys[50]);
    *out = ModuloMultiplication(Data[3],Subkeys[51]);
}
```

5.8.10 Pengurutan Zigzag Sebelum Modifikasi DCT

Berfungsi untuk melakukan pengurutan zigzag terhadap matrik DCT sebelum disisipi data *watermark.*

```
void __fastcall ZigZag_Scan(float *Matrik,float *SigSag,int
lebar,int tinggi)
{
    int i, x, y, P;

    float *Ptr = SigSag;
    for(i=0; i<(tinggi+lebar-2); i++)
    {
        if(i % 2)
        {
            if(i<lebar) x = i; else x = lebar - 1;
            y = i - x;
            do {
               P = y * lebar + x;
               *Ptr = *(Matrik + P);
               x--;
               y++;
               Ptr++;
            } while(!((y > i) || (y > (tinggi - 1))));
        }
```

```
        else {
            if(i<tinggi) y = i; else y = tinggi - 1;
            x = i - y;
            do {
               P = y * lebar + x;
               *Ptr = *(Matrik + P);
               y--;
               x++;
               Ptr++;
            } while(!((x > i) || (x > (lebar - 1))));
        }
    }
}
```

5.8.11 Pengurutan Zigzag Setelah Modifikasi DCT

Digunakan untuk melakukan pengurutan zigzag terhadap matrik DCT setelah disisipi data *watermark*.

```
void __fastcall ZigZag_Ulang(float *Matrik,float *SigSag,int
lebar,int tinggi)
{
    int i, x, y, P;

    float *Ptr = SigSag;
    for(i=0; i<(tinggi+lebar-2); i++)
    {
        if(i % 2)
        {
            if(i<lebar) x = i; else x = lebar - 1;
            y = i - x;
            do {
               P = y * lebar + x;
               *(Matrik + P) = *Ptr;
               x--;
               y++;
               Ptr++;
            } while(!((y > i) || (y > (tinggi - 1))));
        }
```

```
        else {
            if(i<tinggi) y = i; else y = tinggi - 1;
            x = i - y;
            do {
               P = y * lebar + x;
               *(Matrik + P) = *Ptr;
               y--;
               x++;
               Ptr++;
            } while(!((x > i) || (x > (lebar - 1))));
        }
    }
}
```

5.8.12 Subkey Dekripsi IDEA

Digunakan untuk menghasilkan *subkey-subkey* dekripsi di mana *subkey-subkey* enkripsi merupakan parameternya.

```
void __fastcall IDEADecSubkeys(IDEAkey EncSubkeys, IDEAkey
DecSubkeys)
{
    uint16 j, Temp[3], T[52], *Data = T + KEYLEN;

    Temp[0] = inverse(*EncSubkeys++);
    Temp[1] = (uint16)(-*EncSubkeys++);
    Temp[2] = (uint16)(-*EncSubkeys++);
    *--Data = inverse(*EncSubkeys++);
    *--Data = Temp[2];
    *--Data = Temp[1];
    *--Data = Temp[0];

    for (j = 1; j < ROUNDS; j++)
    {
       Temp[0] = *EncSubkeys++;
       *--Data = *EncSubkeys++;
       *--Data = Temp[0];

       Temp[0] = inverse(*EncSubkeys++);
       Temp[1] = (uint16)(-*EncSubkeys++);
       Temp[2] = (uint16)(-*EncSubkeys++);
       *--Data = inverse(*EncSubkeys++);
       *--Data = Temp[2];
       *--Data = Temp[1];
       *--Data = Temp[0];
    }
    Temp[0] = *EncSubkeys++;
    *--Data = *EncSubkeys++;
    *--Data = Temp[0];
```

```
    Temp[0] = inverse(*EncSubkeys++);
    Temp[1] = (uint16)(-*EncSubkeys++);
    Temp[2] = (uint16)(-*EncSubkeys++);
    *--Data = inverse(*EncSubkeys++);
    *--Data = Temp[2];
    *--Data = Temp[1];
    *--Data = Temp[0];

    /* Copy and destroy temp copy */
    for (j = 0, Data = T; j<KEYLEN; j++)
    {
        *DecSubkeys++ = (uint16)(*Data);
        *Data++ = 0;
    }
}
```

5.8.13 Transformasi Invers

Invers DCT dilakukan dengan proses kebalikan dari DCT. Pertama nilai-nilai DCT dikalikan dengan *CosMatrik* $N \times N$, hasil perkalian ini dikalikan lagi dengan *CosTrMatrik*.

```
void __fastcall InvCosTransform(float *src, unsigned char *dst)
{
    float *Buffer;
    Buffer = new float[MatSize*MatSize];
    if(!Buffer)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1);  /* terminate program if out of memory */
    }
    float *wdst = Buffer, *wbit, *wdst2, *wcos, *wf, sum;
    int p, q, r;

    for(p = 0; p < MatSize; wdst++, p++)
    {
        wbit = src;
        wdst2 = wdst;

        for(q = 0; q < MatSize; wdst2 += MatSize, q++)
        {
            wcos = CosMatrik + p;
            sum = 0.0;

            for(r = 0 ; r < MatSize; wcos += MatSize, r++)
                sum += (*wcos * *(wbit++));
            *wdst2 = sum;
        }
    }
    unsigned char *wres = dst;
    for(p = 0; p < MatSize; wres++, p++)
    {
```

```
            wcos = CosTrMatrik;
            unsigned char *wdst2 = wres;

            for(q = 0; q < MatSize; wdst2 += MatSize, q++)
            {
                wf = Buffer + p;
                sum = 0.0;

                for(r = 0; r < MatSize; wf += MatSize, r++)
                   sum += (*wf * *(wcos++));

                int res = sum;
                *wdst2 = (res < 0) ? (unsigned char) 0 : (res > 255) ?
                             (unsigned char) 255: (unsigned char) res;
            }
        }
    }
```

5.8.14 Menghitung PSNR

Digunakan untuk menentukan kualitas citra, semakin besar nilai PSNR berarti penyisipan *watermark* ke dalam citra tidak menyebabkan penurunan kualitas citra secara drastis. Sebaliknya jika nilai PSNR semakin kecil maka pada citra terjadi penurunan kualitas citra secara drastis.

```
float __fastcall TMDI_Child::GetPSNR()
{
    unsigned char g = 255;
    char *SInput;
    char *SWatermark;

    // hitung sample variance
    float SVariance = 0.0;
    for (int m = 0; m < ImageInput->Picture->Bitmap->Height; m++)
    {
      SInput = (char*)ImageInput->Picture->Bitmap->ScanLine[m];
      SWatermark = (char*)ImageWatermark->Picture->
                   Bitmap->ScanLine[m];
      for(int n = 0; n < ImageWatermark->Picture->
         Bitmap->Height; n++)
      {
         SVariance += pow((float)(SWatermark[n] - SInput[n]), 2);
```

```
        }
        SVariance /= (ImageInput->Picture->Bitmap->Width
        *ImageInput->Picture->Bitmap->Height);
    }
    float PSNR = 10 * log10(pow(g, 2) / SVariance);
    return PSNR;
}
```

BAB VI

UJI COBA DAN EVALUASI

BAB VI

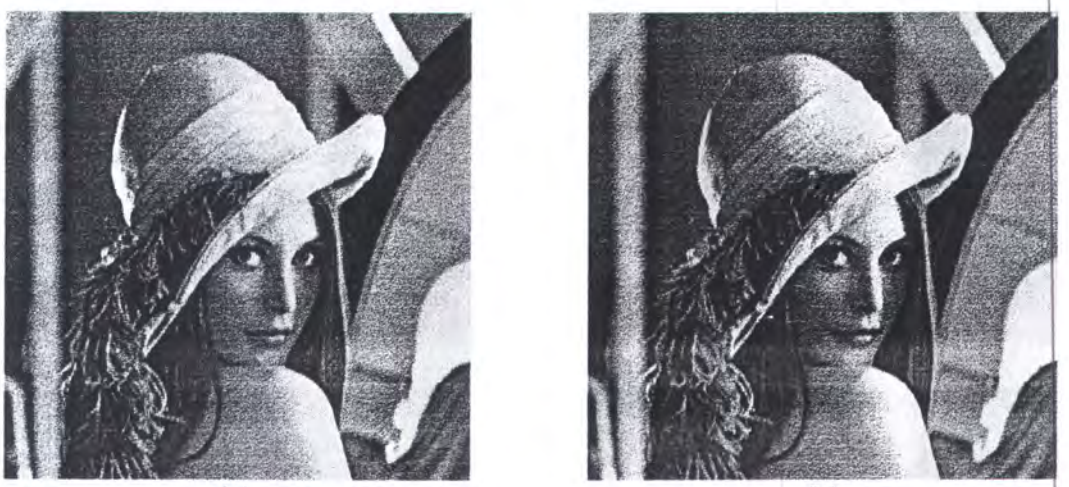
UJI COBA DAN EVALUASI

Uji coba untuk mengetahui sampai sejauh mana unjuk kerja perangkat lunak ini adalah dengan menyisipkan data *watermark* ke dalam citra dengan bermacam-macam nilai energi *watermark* dan berbagai macam jumlah koefisien yang dilewati serta jumlah data *watermark* yang diinputkan. (Gambar 6.1 sampai Gambar 6.16). Masing-masing parameter tadi akan menghasilkan citra yang tersisipi *watermark* dan citra yang tidak tersisipi *watermark* serta terjadi penurunan kualitas citra secara drastis atau tidak drastis setelah tersisipi data *watermark*.

6.1 Uji Coba Perangkat Lunak

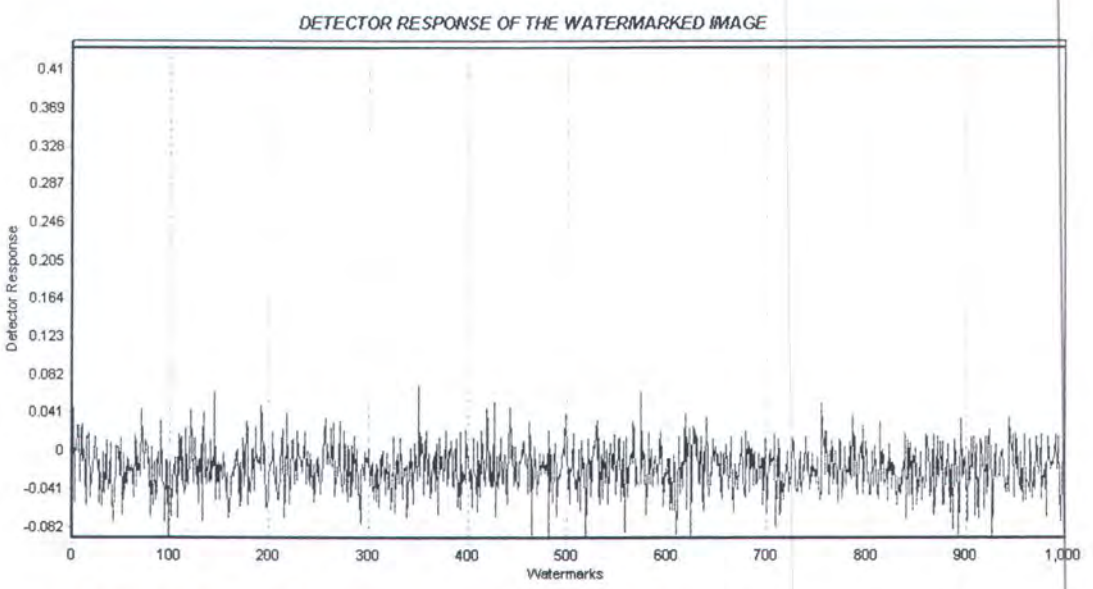
Berikut akan ditampilkan citra-citra analisa beserta grafik pendeteksian keberadaan data *watermark* yang dilakukan terhadap file citra Lenna256.BMP dan Ninna256.BMP. Dari (Gambar 6.1 sampai Gambar 6.16) dapat dilihat citra asal dan citra yang telah tersisipi data *watermark*. Citra *watermark* diperoleh karena perubahan besar energi *watermark*, jumlah data *watermark* dan jumlah koefisien yang dilewati. Semakin tinggi energi untuk *watermark*, maka *watermark* tersebut akan semakin kuat dan tampak pada citra, misalnya pada Gambar 6.7 bagian kiri atas. Grafik yang ditampilkan digunakan untuk mendeteksi keberadaan data *watermark*. Jika nilai *threshold* lebih rendah dari nilai korelasi, maka dapat

disimpulkan bahwa citra tersisipi data *watermark*. Sebaliknya jika nilai *threshold* lebih tinggi dari nilai korelasi, maka citra tidak tersisipi data *watermark*.



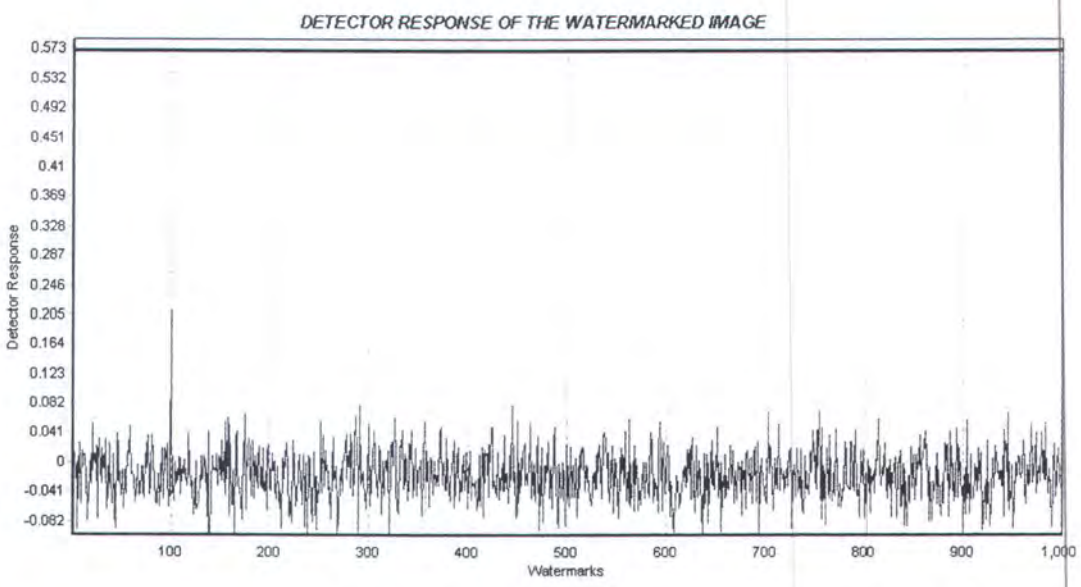
(a) (b)

Gambar 6.1 (a) Citra asal, (b) Citra watermark dengan α = 0, n = 16000, k = 16000



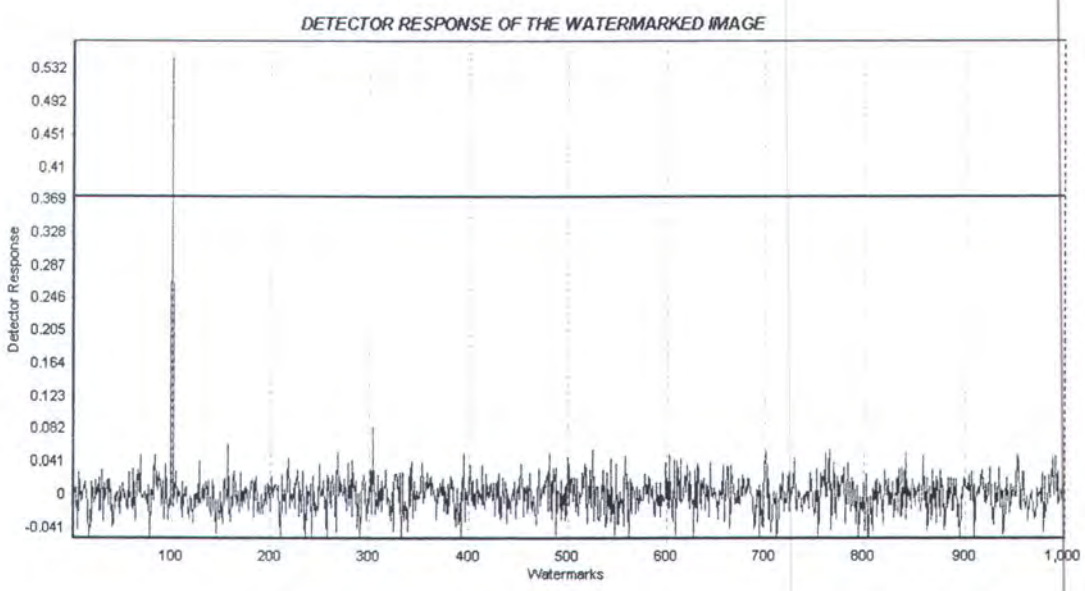
Gambar 6.2 Grafik pendeteksian keberadaan watermark dengan α = 0, n = 16000, k = 16000

Gambar 6.3 Citra watermark dengan α = 0.1, n = 10000, k = 17500



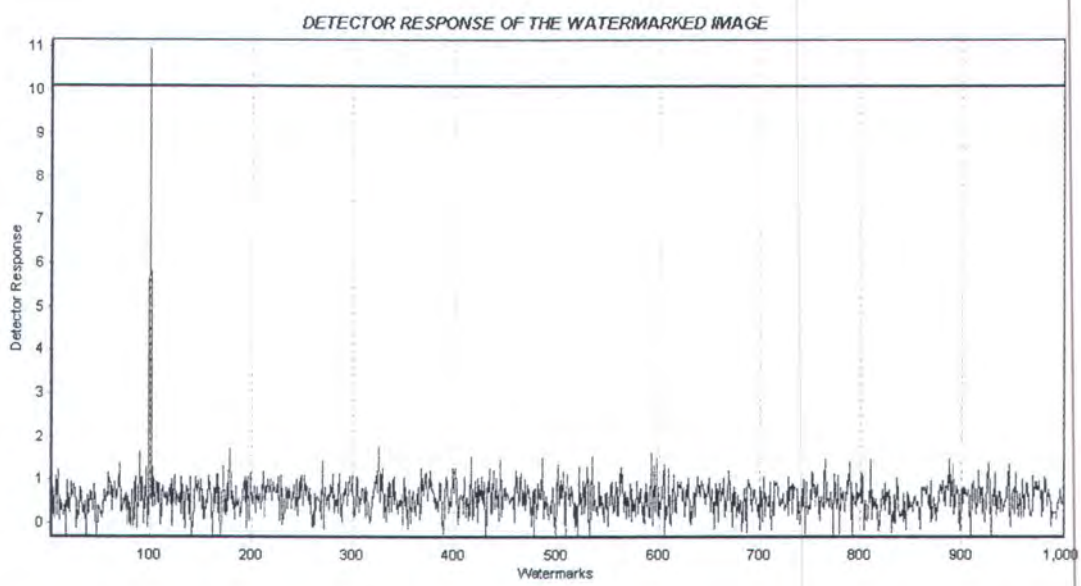
Gambar 6.4 Grafik pendeteksian keberadaan watermark dengan α = 0.1, n = 10000, k = 17500

Gambar 6.5 Citra Watermark dengan α = 0.2, n = 25000, k = 12000



Gambar 6.6 Grafik pendeteksian keberadaan watermark dengan α = 0.2, n =25000, k = 12000

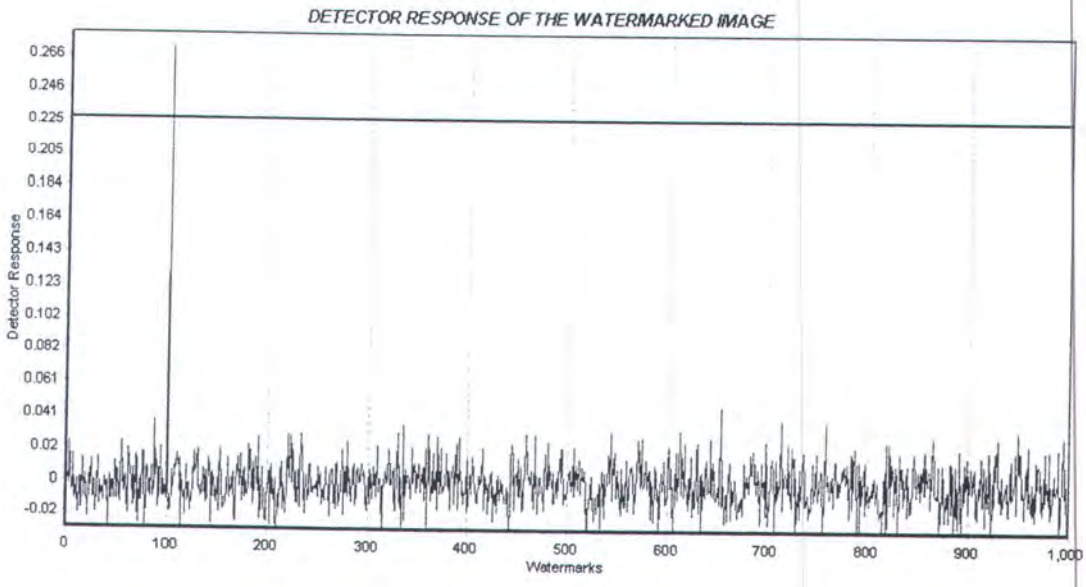
Gambar 6.7 Citra Watermark dengan α = 1, n = 1000, k = 4000



Gambar 6.8 Grafik pendeteksian keberadaan watermark dengan α = 1, n = 1000, k = 4000

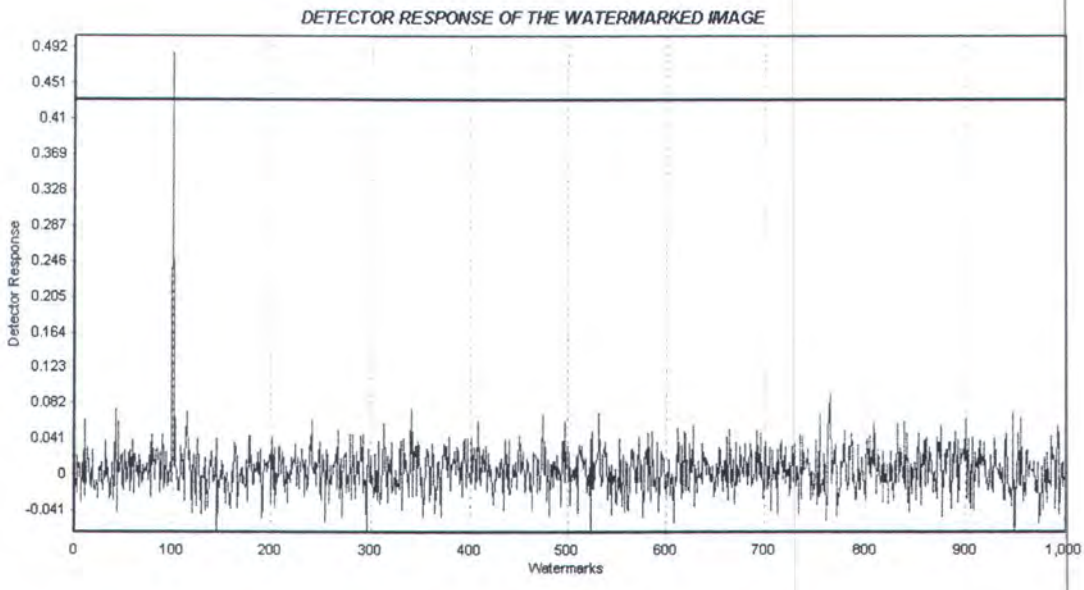
(a) (b)

Gambar 6.9 (a) Citra asal, (b) Citra watermark dengan α = 0.175, n = 16000, k = 20000



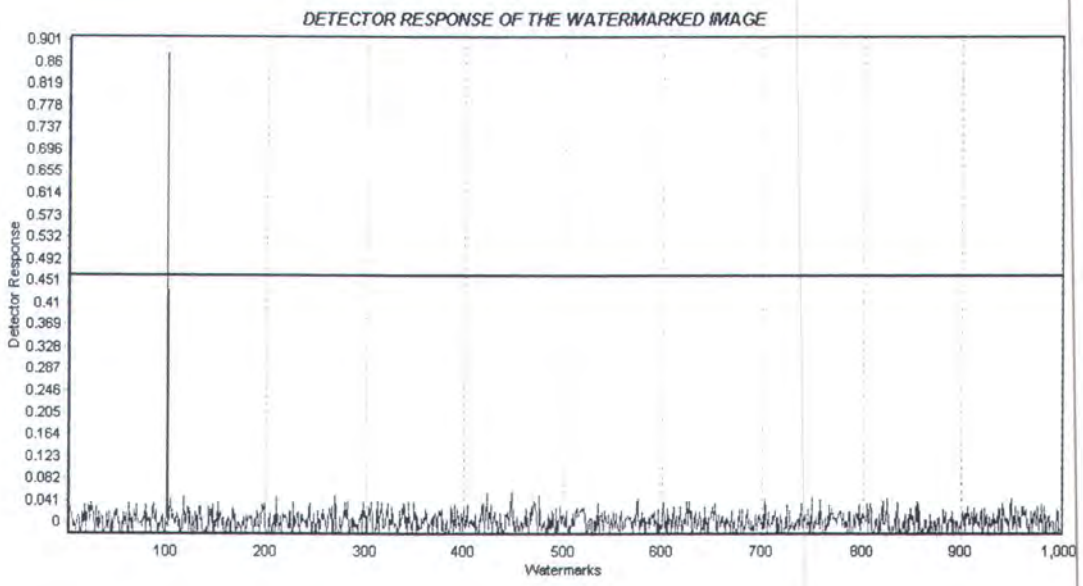
Gambar 6.10 Grafik pendeteksian keberadaan watermark dengan α = 0.175, n = 16000, k = 20000

Gambar 6.11 Citra Watermark dengan α = 0.373, n = 3000, k = 35000



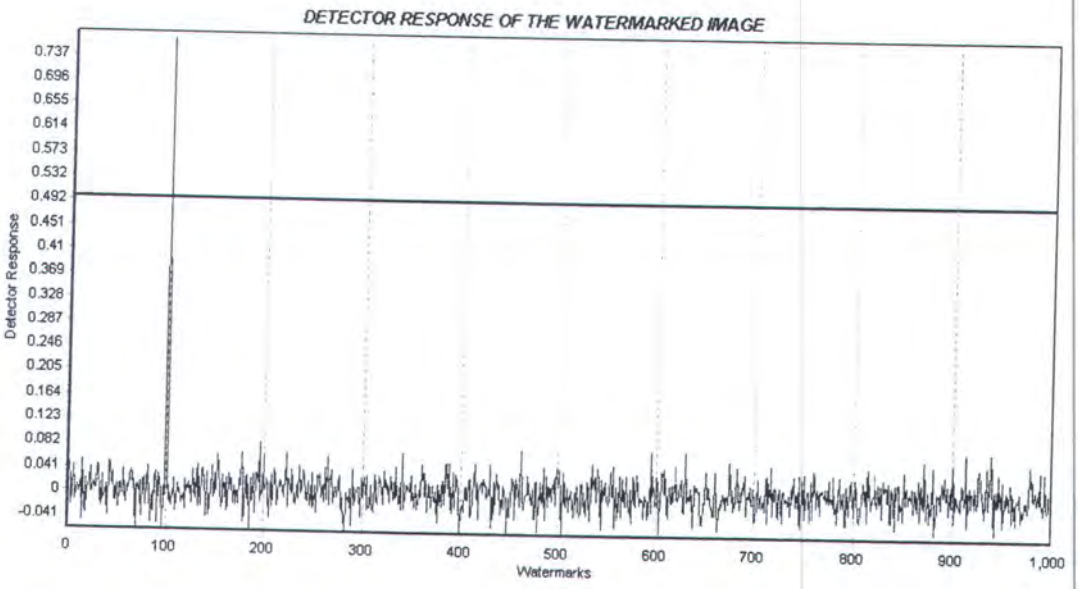
Gambar 6.12 Grafik pendeteksian keberadaan watermark dengan α = 0.373, n = 3000, k = 35000

Gambar 6.13 Citra Watermark dengan α = 0.925, n = 3000, k = 45000



Gambar 6.14 Grafik pendeteksian keberadaan watermark dengan α = 0.925, n = 3000, k = 45000

Gambar 6.15 Citra Watermark dengan α = 0.625, n = 3000, k = 35000



Gambar 6.16 Grafik pendeteksian keberadaan watermark dengan α = 0.625, n = 3000, k = 35000

Gambar 6.17 Citra Watermark dengan α = 0.2, n = 16000, k = 16000

Gambar 6.18 Citra Watermark dengan α = 0.2, n = 16000, k = 40000

6.2 Analisa Hasil

Dari uji coba yang dilakukan, analisa yang diperoleh ditunjukkan oleh tabel berikut ini. Tabel 6.1 dengan energi *watermark* (α) yang sama menunjukkan perbedaan PSNR dan *Sample Variance* yang terjadi bila dilakukan pengubahan jumlah data *watermark* dan jumlah koefisien yang dilewati (k). Dari tabel ini dapat dilihat bahwa untuk energi *watermark* (α) yang sama, semakin besar jumlah data *watermark* (n) yang disisipkan ke dalam citra dan semakin besar jumlah koefisien yang dilewati (k) maka nilai PSNR semakin besar sedangkan nilai *Sample Variance* semakin kecil. Kualitas citra dapat dilihat pada halaman sebelumnya.

Untuk mengevaluasi kualitas citra ini dipergunakan *peak-to-peak signal-to-noise ration* (PSNR).

$$PSNR = 10\ Log_{10}\left[\frac{g^2}{\sigma_r^2}\right]\ dB$$

g = perbedaan antara *gray level* maksimum dengan minimum. *Gray level* 256 ; nilai g = 255

σ_r^2 = *sample variance* dari kesalahan proses *watermark* $r(m,n)$

$r(m,n) = x'(m,n) - x(m,n)$

$x(m,n)$ menyatakan nilai piksel citra asal dan $x'(m,n)$ menyatakan nilai piksel citra *watermark* pada posisi (m,n).

Untuk menghitung kesalahan yang digunakan di sini adalah *Sample Variance* yang dinyatakan dengan persamaan :

$$\sigma_r^2 = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} r^2(m,n)$$

Berikut ini adalah hasil evaluasi PSNR dan *Sample Variance* untuk nilai-nilai ***k*** dan ***n*** yang berbeda dihitung untuk citra standard "Lenna" dan "Ninna" (α = 0.2).

Citra	Jumlah Watermark	Jumlah Koefisien Terlewati	Sample Variance	PSNR
Lenna	1000	1000	0.008682	68.7444
	8000	8000	0.00325	73.0118
	16000	16000	0.002441	74.2544
Ninna	1000	1000	0.003143	73.1569
	8000	8000	0.002289	74.5346
	16000	16000	0.001862	75.4322

Tabel 6.1 Evaluasi PSNR dan *Sample Variance* untuk nilai-nilai *k* dan *n* yang berbeda dihitung untuk citra standard "Lenna" dan "Ninna" (α = 0.2)

Pada Tabel 6.1 terlihat perbedaan antara citra Lenna dan citra Ninna dengan jumlah *watermark* dan jumlah koefisien terlewati yang sama, *sample variance* untuk citra Lenna lebih besar dari citra Ninna. Hal ini disebabkan karena citra Ninna lebih terang dari citra Lenna.

BAB VII

KESIMPULAN DAN SARAN

BAB VII

KESIMPULAN DAN SARAN

DCT adalah transformasi yang mengolah sinyal tiga dimensi yang digambar pada sumbu-sumbu X, Y, Z. Sumbu-sumbu X dan Y menunjukkan frekuensi dari sinyal dalam dua dimensi yang berbeda. Amplitudo dari sinyal adalah nilai dari piksel pada titik di layar. Gambar grafik yang ditampilkan di layar dapat dianggap sebagai sinyal tiga dimensi yang kompleks dengan nilai sumbu Z ditunjukkan oleh warna pada layar pada titik yang bersangkutan.

Dalam matrik DCT $N \times N$, semua elemen dalam baris ke 0 memiliki komponen frekuensi nol pada arah yang sama. Semua elemen dalam kolom ke 0 memiliki komponen frekuensi nol dalam arah yang lain. Sejalan dengan bergesernya baris dan kolom dari titik asal (0,0), koefisien-koefisien dalam matrik DCT yang ditransformasikan mulai menunjukkan kenaikan frekuensi, dengan frekuensi tertinggi pada koordinat $(N-1, N-1)$. Seperti pada Gambar 6.7, citra mengalami penurunan kualitas secara drastis, karena penyisipan data *watermark* ke dalam citra dilakukan pada koefisien-koefisien yang memberi kontribusi sajian yang sangat berarti pada citra. Sedangkan Gambar 6.11 terlihat seperti citra asal, karena penyisipan data *watermark* ke dalam citra dilakukan pada koefisien-koefisien yang membawa informasi yang tidak sepenting koefisien-koefisien sebelumnya.

Ini sangatlah berarti, karena citra grafis pada layar dibentuk dari informasi yang berfrekuensi rendah. Sejalan dengan bergesernya baris dan kolom dari titik asal (0,0), koefisien-koefisien tidak saja cenderung memiliki nilai yang lebih rendah melainkan mereka juga membawa informasi yang tidak sepenting koefisien-koefisien sebelumnya. Misalnya pada Gambar 6.5 dan Gambar 6.7 .Pada Gambar 6.5 penyisipan *watermark* dilakukan pada koefisien-koefisien yang berfrekuensi tinggi sehingga tidak menyebabkan penurunan kualitas citra secara drastis. Sedangkan pada Gambar 6.7 penyisipan *watermark* dilakukan pada koefisien-koefisien yang berfrekuensi rendah sehingga citra mengalami penurunan kualitas secara drastis. Jika Gambar 6.5 dilihat dari hasil printer perubahan pada pojok kiri atas tidak terlihat jelas. Sedangkan jika dilihat pada layar monitor dapat terlihat jelas.

Sangat tepat jika metode *watermarking* digunakan untuk melakukan proteksi data citra, karena metode ini memanfaatkan sifat pemampatan energi pada transformasi DCT.

Proteksi citra dilakukan dengan menggunakan data *watermark* yang disisipkan pada koefisien DCT yang terpilih sehingga dihasilkan citra seperti yang terlihat pada Gambar 6.9. Selanjutnya dilakukan pendeteksian keberadaan data *watermark* di dalam citra seperti yang tampak pada Gambar 6.10.

Faktor-faktor yang berpengaruh dalam proses proteksi data citra ini adalah energi *watermark* (α) dan banyaknya koefisien yang dilewati (k) serta jumlah data *watermark* (n). Jika energi *watermark* tetap, sedangkan jumlah data *watermark* lebih kecil atau sama dengan jumlah koefisien yang dilewati maka PSNR semakin

besar dan *Sample Variance* semakin kecil. Ini dapat dilihat dari hasil komputasi dari program yang penulis buat sebagai berikut :

Misalnya pada Gambar 6.17 dengan $\alpha = 0.2$, $n = 16000$, $k = 16000$, diperoleh *Sample Variance* = 0.0025482466 dan PSNR = 74.0683898926. Sedangkan pada Gambar 6.18 dengan $\alpha = 0.2$, $n = 16000$, $k = 40000$, diperoleh *Sample Variance* = 0.0020141895 dan PSNR = 75.0897979736. Semakin banyak koefisien-koefisien yang terlewati maka penyisipan data *watermark* ke dalam citra tidak menyebabkan penurunan kualitas citra secara drastis.

Simulasi yang dilakukan pada citra-citra sederhana menunjukkan bahwa penyisipan data *watermark* ke dalam citra tidak menurunkan kualitas citra secara drastis. Seperti pada Gambar 6.9 (b), Gambar 6.11, Gambar 6.13, Gambar 6.15.

Pengembangan lebih lanjut penelitian dapat dititik-beratkan pada pembuatan prosedur koreksi kesalahan sehingga *watermark* dapat digunakan untuk memperoleh data asli walaupun telah dilakukan proses *filtering*, kompresi JPEG bahkan *multiple watermarking* pada citra.

DAFTAR PUSTAKA

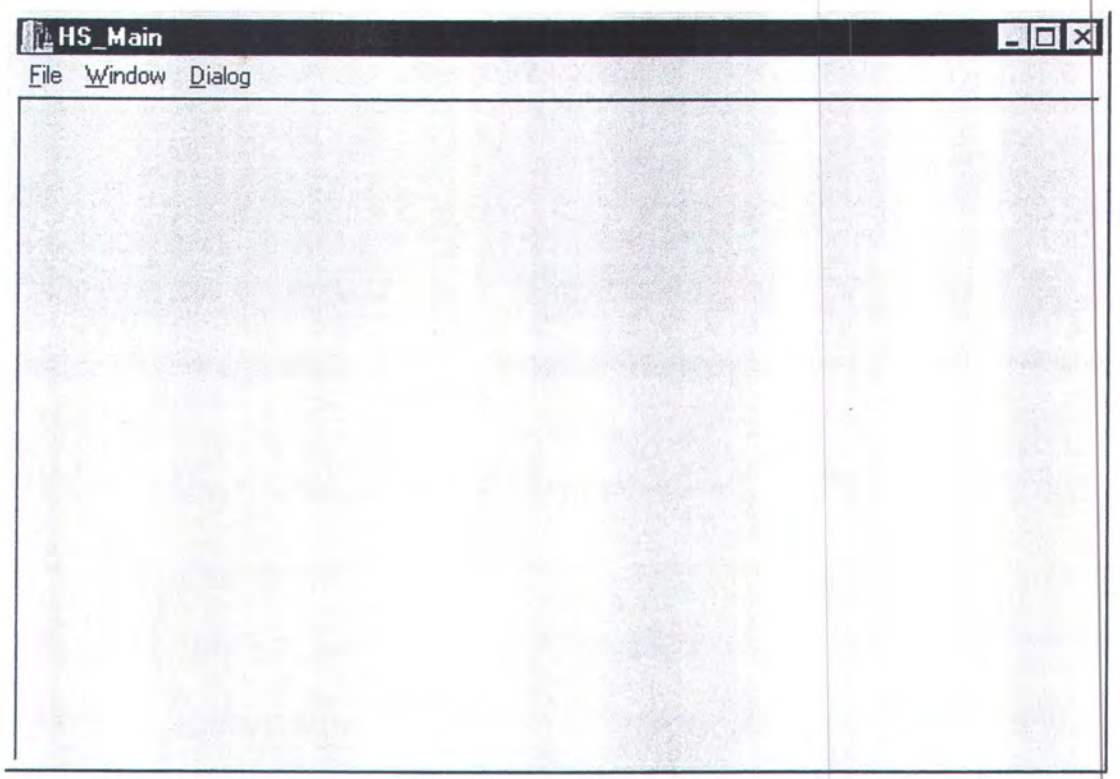
DAFTAR PUSTAKA

1. A. Piva, M. Barni, F. Bartolini, V.Cappellini, "*A DCT-Domain System for Robust Image Watermarking*", Firenze, Dipartimento di Ingegneria Elettronica, Universita di Firenze,1998
2. A. Piva, M. Barni, F. Bartolini, V.Cappellini, "*Threshold Selection for Correlation-Based Watermark Detection*", Dipartimento di Ingegneria Elettronica, Universita di Firenze
3. I.Nyoman Suyoga, "*Rancang Bangun Perangkat Lunak Kompresi Dan Dekompresi Dengan Menggunakan Metode Transformasi Cosinus Diskrit*", TEKNIK INFORMATIKA, ITS di Surabaya,1995
4. A. Piva, M. Barni, F. Bartolini, V. Cappellini, "*Image Watermarking for Secure Transmission over Public Network* ", Firenze, Dipartimento di Ingegneria Elettronica, Universita di Firenze, 1982
5. Schneier Bruce, "*Applied Cryptography Protocols, Algorithms, and Source Code in C* ", John Willey & Sons,1994
6. J. Zhao and E. Koch, Proceeding of the Int. Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Technology, "*Embedding robust labels into images for copyright protection* ", Vienna, August 1995, 242-252.

LAMPIRAN

LAMPIRAN

MENU PERANGKAT LUNAK



Gambar lampiran-1. Menu Perangkat Lunak Proteksi Data Citra

Menu perangkat lunak ini terdiri dari 3 macam yaitu :

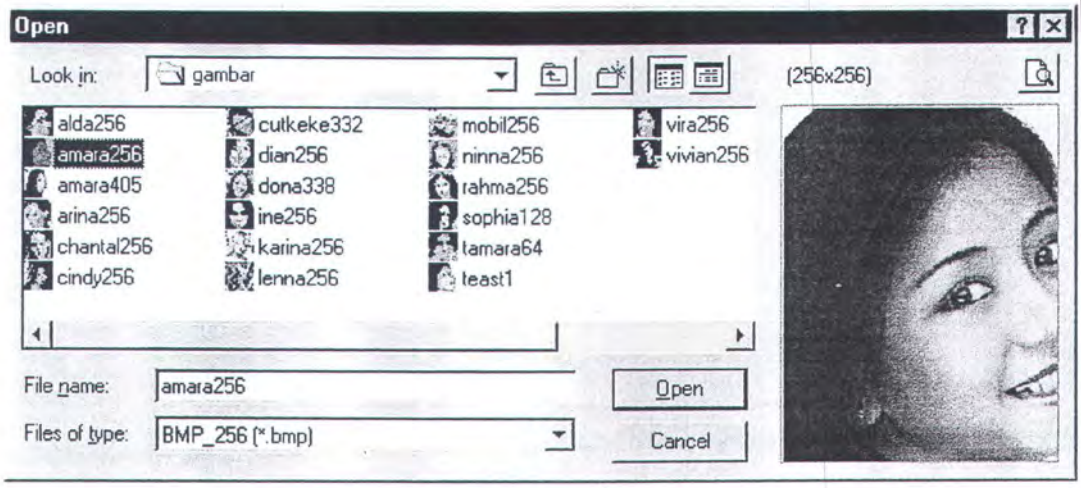
1. File
2. Dialog
3. Window

Menu File mempunyai menu-menu sebagai berikut :

1. **Open**

   Berfungsi untuk memanggil File bitmap yang akan disisipi data *watermark*.

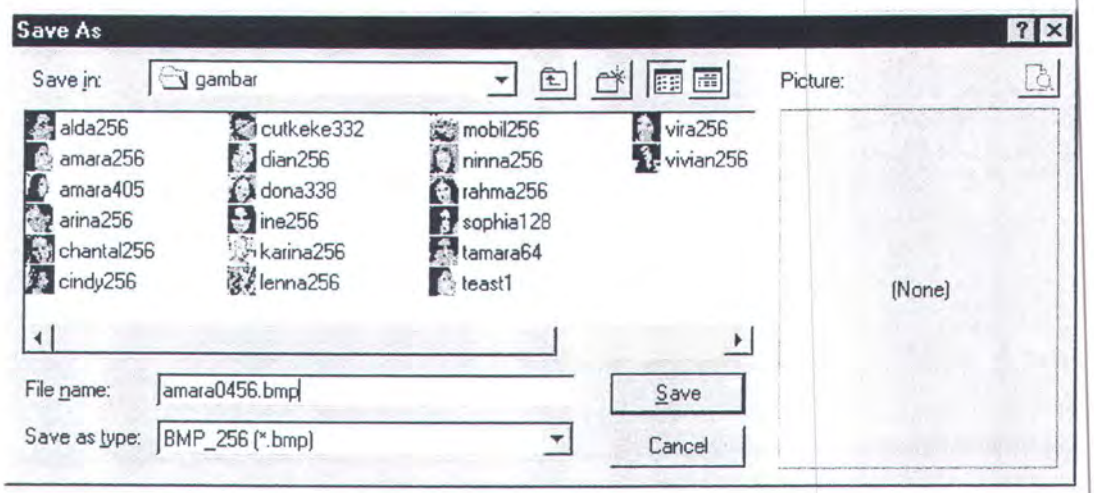
   Dari menu ini akan muncul dialog untuk memasukkan nama file.



Gambar lampiran-2. Dialog untuk memasukkan nama file

2. **Save As**

Berfungsi untuk menyimpan citra yang telah tersisipi data *watermark* ke dalam bentuk File bitmap. Dari menu ini akan muncul dialog untuk memberikan nama file dari citra *watermark* tersebut.



Gambar lampiran-3. Dialog untuk memberikan nama file citra watermark

3. **Close**

Menutup window tampilan citra dan kembali ke window utama.

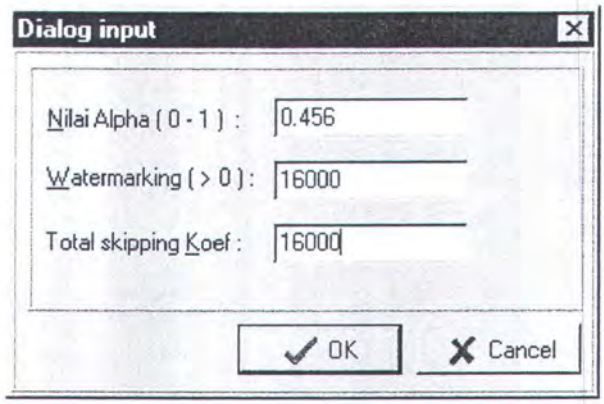
4. **Exit**

Keluar dari perangkat lunak Proteksi Data Citra.

Menu Dialog terdiri dari menu-menu sebagai berikut :

1. **Parameter**

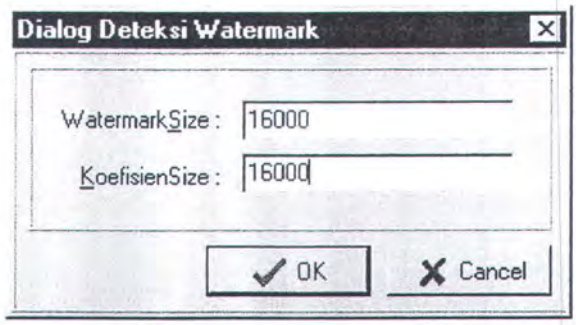
   Berfungsi untuk memasukkan nilai energi *watermark*, jumlah data *watermark*, jumlah koefisien yang terlewati. Dari menu ini akan muncul dialog input.



Gambar lampiran-4. Dialog untuk memasukkan parameter-parameter

2. **Deteksi**

   Berfungsi untuk mendeteksi keberadaan data *watermark* yang telah disisipkan ke dalam citra, dengan cara memasukkan jumlah data *watermark* yang disisipkan dan jumlah koefisien yang terlewati. Dari menu ini akan muncul dialog deteksi *watermark*.



Gambar lampiran-5. Dialog untuk memasukkan nilai Password

Menu berikutnya adalah Window yang berkaitan dengan modifikasi penampilan window-window di layar. Di antaranya ada menu Tile, sub menu Vertical, sub menu Horizontal, dan menu Cascade.