GOVP1200211601

정 책 연 구 사 업

암호화된 범죄집단의 송수신 메시지 패턴인식 및 감시기술 개발

(Development of Encrypted Message Pattern Recognition Technology of Criminal Groups and Message Monitoring Technology)

과 학 기 술 부

최종보고서

관리번호		기술분류	
과 제 명	암호화된 범죄집단의 송수신 메시지 패턴 인식 및 감시기술 개발 (영문) Development of Encrypted Message Pattern Recognition Technology of Criminal Groups and Message Monitoring Technology		
주 관 연구기관	기 관 명	소 재 지	대 표
	상명대학교	서울시 종로구 홍지동 7번지	방 정 복
주관연구 책 임 자	성 명	소속 및 부서	전 공
	최 종 욱	상명대 정보통신학부	정 보 학
연구기간	2000년 11월 14일 ~ 2001년 11월 13일 (12개월)		
연 구 비	일금 89,000,000 (정부 : 69,000 민간 : 20,000)천원정		
참여연구원	8 명(책임 : 1명, 연구원 : 1명, 연구보조원 : 5명, 보조원 : 1명)		

2000년도 정책 연구개발사업에 의하여 수행중인 연구과제(사업)의 최종보고서를 붙임과 같이 제출합니다.

o 첨부 : 최종보고서 10부

2001년 10월 12일

주관연구책임자 최 종 욱

주관연구기관장 서 명

과학기술부장관 귀 하

암호화된 범죄 집단의 송수신 메시지의 패턴 인식 및 감시 기술개발 에 관한 정책연구사업의 최종보고서를 별첨과 같이 제출합니다.

2001년 10월 12일

주관연구책임자 최 종 욱 (인)

주관연구기관장 서 명 (직인)

제 출 문

과학기술부장관 귀하

본 보고서를 "암호화된 범죄집단의 송수신 메시지 패턴인식 및 감시기술 개발" 최종보고서로 제출합니다.

2001년 10월 12일

○ 주관연구기관명 : 상 명 대 학 교

○ 연 구 기 간 : 00.10.14-01.11.13

○ 주관연구책임자 : 최 종 욱

○ 참여 연구원

* 연구원 : 김대수
* 연구원 : 이진영
* 연구원 : 이한호
* 연구원 : 최기철
* 연구원 : 유명주
* 연구원 : 이용남
* 보조원 : 정길호

요 약 문 (국문)

1. 제목

암호화된 범죄집단의 송수신 메시지 패턴인식 및 감시기술 개발

2. 연구개발의 목적 및 중요성

군사적인 용도로 사용되던 암호화 알고리즘이 인터넷의 확산으로 일반인들과 상업용으로 사용됨에 따라 기존의 감청에 의존하던 국가기관의 수사와 법 집행이 상당한 어려움에 봉착하였다. 특히 테러와 마약, 유괴와 청소년 매춘, 탈세와 자금 세탁 등의 범죄행위가 인터넷과 암호화 기술을 이용하여 손쉽게 국경을 넘나들고 국가기관의 수사망을 파할 수 있게되어 암호화 해독기술이 국가의 안전과 공공의 안전을 위해 필요하게되었다.

본 연구에서는 네트워크에서 사용되는 암호 메시지로부터 그 암호문이 어떤 알고리즘을 사용하는지를 판단하고 내부자의 기밀 정보 유출을 파악할 수 있는 네트워크 감시 기술을 개발하고자 하였다. 이는 비록 암호화된 문서를 해독하기는 어렵지만 그 문서에 대한 정보만이라도 추출함으로서 범죄행위의 적발과 수사에 도움이 될 수 있기 때문이다. 특히 네트워크를 감시하여 일단 암호화된 문서를 가려내고 이를 사용하는 사용자들을 집중적으로 할 수 있다면 이는 상당한 도움이 될 것으로 기대한다.

3. 연구개발의 내용 및 범위

마약, 테러, 유괴 등의 범죄에 가담한 집단의 통신데이터나 저장 데이터를 해독하기 위해서는 (1) 우선 평문과 암호문의 구분을 정확히 할 수 있어야 하며 (2) 암호문의 경우 사용된 암호화 알고리즘을 판별하고 (3) 가능하다면 작성된 문서의 종류를 구분할 수 있어야 한다는 것이 본 연구의 기술적인 목표로 수립되었다.

암호문과 평문의 판별에 있어서는 문서가 300Byte이상이 되면 95%이상의 정확도를 가지며 암호화에 사용된 알고리즘의 인식률이 80%이상이 되도록 목표를 수립하였다. 네트워크 감시는 300개 이상의 노드를 가진 네트워크를 패킷분석으로 실시간으로 감시하고 불법 송수신시 5분 이내에 판별이 가능해야 한다는 기술적인 목표를 수립하였다.

4. 연구개발 결과

평문과 암호문의 경우, 고전적인 문자 분포 통계 데이터를 사용하여서도 가능하지

만 주파수 공간에 투영하였을 경우의 correlation 측정으로도 가능해진다. 본 연구에서 시도된 correlation 측정법은 메시지의 길이가 300Byte이상인 경우 100%의 정확도를 보였으며 300Byte이하인 경우에도 100Byte 이상이면 97%이상의 정확도를 보였다.

본 연구의 가장 중요한 목표는 암호문이 경우 패턴 분석으로부터 사용된 암호화 알고리즘을 판정(Identify)하는 것이다. 본 연구의 사전 연구에서 대칭 키 알고리즘인 DES, IDEA를 비교하였을 경우 recurrence Plot이나 return map을 통해서 확연하게 다른 패턴이 나타났기 때문이다. 이는 대단히 중요한 결과로서 현재까지 대부분의 암호 분석 연구들이 암호 키를 찾아내기 위한 무차별 공격(Brute-Force Attack)에 매달리고 있다는 점을 생각하면 대단히 획기적인 연구결과가 얻어질 수 있는 가능성이 있었다. 그러나 되풀이되는 실험을 통해 사전 연구의 결과가 잘못되었다는 점을 확인하였다. 이는 프로그램의 오류로서 결과적으로 암호화 알고리즘들 간의 패턴 구분은 암호문에서는 불가능하다는 것이 본 연구의 결론이다.

이처럼 암호화 알고리즘간의 차이가 생성된 암호문을 통해서 관찰되지 않는 것은 기본적으로 모든 암호문이 Random Number를 생성하도록 구조가 만들어져 있기 때문이다. 첫번째 시도로서 약 52,000개의 일반 데이터를 암호화하고 이를 8Byte 블럭 단위로 복호화 하여 생성된 시계열 데이터를 return map이나 recurrence plot으로 하였으나 알고리즘별 차이는 관찰되지 않았다. 이를 바꾸어 생성된 블록에서 1Byte 단위로 복호화하여 return map과 recurrence plot을 그려보았으나 역시 패턴은 나타나지 않았다. Hurst Index 측정 등의 여러 가지 시도를 통해서도 같은 결과가 얻어졌으면 결론적으로 대칭형 키에서는 암호화된 문장으로부터 사용된 알고리즘을 구분하는 것이 불가능하다는 결론에 도달하게 되었다.

다음으로는 암호문이라고 판정되는 경우, 그 암호문이 원래 어떤 파일로 만들어졌는지를 구분하기 위한 연구가 진행되었다. 이외로 암호화 알고리즘에 대해 원시 파일의 파일 형식에 따라 각기 다른 패턴이 생성됨을 알 수 있었다. 즉, 암호화된 문장으로부터 그 원래 파일이 MS Word, Power Point, 아래 한글, 엑셀, Zip압축, Jpeg압축 등의 구분이 가능함을 알게 되었다.

네트워크 감시 기능으로는 근거리 통신망 내 모든 TCP/IP 패킷을 실시간으로 분석하여 암호화된 메시지가 전송될 경우 송,수신 IP주소를 리포팅 하도록 하였다. 그리고 특정 IP주소에 대한 감시 기능을 구현하였다. 즉, 감시관리자가 지정한 IP주소에 대하여 모든 송수신 패킷을 감시하여 실시간으로 송수신되는 메시지의 패턴을 리포팅 하도록 하였다. 이는 주로 현재 전송되고 있는 메시지가 평문인지 암호문인지 그리고 평문이면 어떤 종류의 평문인지 등의 정보를 보고하도록 하였다.

5. 검토결과

평문/암호문의 패턴 인식을 위해 통계적인 방법과 카오스 이론을 적용한 방법 그리고 시스템 상사를 이용한 신호처리 방법을 사용하였으나 바이트 단위의 ASCII값 분포를 이용한 통계적인 방법이 현실적(계산량)으로 가장 적합하다고 판단된다.

그리고 암호알고리즘 인식의 경우 암호알고리즘(대칭형키 방식과 해쉬알고리즘)간 명확한 패턴의 차이를 사전연구에서 사용된 모든 방식에서 찾아내지를 못했다. 즉, 기존의 암호알고리즘에서는 주기적인 특성이 거의 없다는 것이다. 하지만 부수적으로 암호문이 어떤 평문에 의해 만들어졌는지는 추정이 가능한 결과를 확인하였다.

6. 보완점

- 비대칭형 암호알고리즘 패턴을 추가 확장
- MAC 계층의 패킷 캡쳐링 모듈의 완전 JAVA화

ABSTRACT

1. Title

Development of Pattern Recognition Technology for Encrypted Message of Criminal Group and Network Monitoring Technology

2. Purpose and Importance of R&D

As encryption technologies that have been used for military purposes have become widely utilized and popularized by people-on the street and business transactions, government organizations have been confronted difficulties in criminal investigation and law enforcement who have relied for long on wiretapping, an analog technology. Especially, socially critical criminal groups such as terror, drug, kidnapping, juvenile prostitution, tax frauds, and money laundry very easily cross national borders with speed through internet and encryption technology. They easily escape out trap of law enforcement organizations using information technology and network, which endangers national safety and public safety.

This research aims at developing a network monitoring technology which can discriminate encryption algorithms employed and identify information leakage. Even though encrypted document cannot be decrypted without encryption key, obtaining information of document might be of a great help to criminal investigation. In special, identification of users of encrypted document and then focusing monitoring activities on the users will drastically reduce time and energy.

3. Scope of Research

To decrypt communication data and stored data of criminals involved in drug, terror, kidnapping, and money laundry, the following should be done: (1) discrimination between plaintext and cipher-text should be possible (2) if the message is encrypted, algorithm of the message should be identified (3) if possible, the file format of encrypted message should be identified.

The target of research in discriminating plain text and cipher text is accurate more than 95% when the text is longer than 300 Byte. Accuracy in identifying encryption algorithm should be higher than 80%. The number of possible

network monitoring should be more than 300 nodes for real-time packet analysis and the time limit should be 5 minute fro reporting to network system administrations.

4. Research Results

In case of discriminating plaintext from cipher text, statistical analysis can provide pretty high accuracy. In this research correlation measurement was employed for the discrimination. The correlation measurement showed 100% accuracy for the messages longer than 300Bytes, higher than 97% for the messages for longer than 100bytes but less than 300 bytes.

The most important target of this research was to identify encryption algorithm employed in message encryption based on pattern analysis. This very ambitious target was established because potentially promising test result was obtained in the previous research that showed significantly different patterns in recurrence plot and return map. The analysis was based on the time-series analysis using fractal analysis technique by mapping encrypted data into time-series domains. The result was very important because it indicates that there might be possibility to find way to distinguish encryption algorithms in fractal space. However, in the repeated experiments it was found that such a significant difference does not exist in the recurrence plot and return map for the encrypted message data. The research team reached a conclusion that the test result obtained in the previous research was wrongly presented and tested in wrong manner.

The reason why difference between algorithms can not be observed is that encryption algorithms basically have smoothing effects for any kind of test message, generating randomly dispersed numbers. In the first stage a large number of dat set, 52000 characters, were tested againt the recurrent plot and return map using 8 bytes in a block. Then, the block was divided into 1 byte character set. In both experiments, no significant difference was observed. Several other indice, such as Hurst Indicator, Fractal dimension, and others were measured but not evidence of patterns was observed. As a conclusion, the pattern observed in previous research was wrong and such a difference in pattern can not be found in fractal world.

Then, a question was raised for the file format before encrytion when the message is classified into encrypted one. Unexpectedly significantly different patterns were statistically observed. That is, from the encrypted message, it can be identified which file format was employed in encryption: MS Word, Power Point, 아래 한글, Excel, Zip compression, Jpeg compression, avi compression and others.

For the network monitoring, all TCP/IP packet data in LAN was analyzed in real-time mode and when the message is encrypted Ip addresses of sender and receiver are reported to system people. That is, for the IP addresses specified by the system monitoring people all packet data are monitored for the senders, receivers and message format. This explains why identification of message format is important to network monitoring.

5. 검토결과

Statistical method and chaos technique were employed to discriminate plaintext from cipher text. In the experiments, it was found that the statistical method based on ASCII distribution is appropriate for practical applications, even though accuracy can be enhanced by combining statistical method and chaos technique.

Discrimination of encryption algorithm is not possible in the fractal space analysis. That is, the time series data obtained from encrypted message does not show any cyclical patterns. However, it is possible to find the file format before the encryption.

6. Future works

-Pattern Analysis of Asymmetric algorithms

-Complete JAVA programming for packet capturing module at MAC layer.

목 차

제 5장 결론

1. 연구의 결론
2. Contribution
3. 기술의 활용

CONTENT

그 림 목 차

표 목 차

제 1장 서 론

1. 연구의 필요성

본 연구의 목적은 네트워크에서 사용되는 메시지로부터 그 메시지가 암호문인지 여부와 암호문인 경우 어떤 암호화 알고리즘을 사용하는 지를 판정하고 내부자의 기밀유출을 감시하고 방지할 수 있는 네트워크 감시 기술을 개발하는 것이다. 이는 암호문을 해독하는 Cracking 기술의 첫 단계로서 일단 암호화하여 전송된다고 의심되는 메시지를 별도로 분류하고 이 별도의 분류된 메시지로부터 암호화에 사용된 알고리즘을 구분해 내고 정확하지는 않지만 내용을 추측해내려는 시도이다. 이러한 기술은 군사적인 목적을 위해서 개발된 암호화 알고리즘이 인터넷의 보편적 사용에 의한 상업적인 용도의 증가로 국가에서 감시하여야할 테러, 마약, 청소년 유괴, 조직 폭력단 등의 범죄집단 동태 파악과 증거 수집에 효과적으로 대처하기 어렵게 되었기 때문에 국가기관에 의해서 반드시 개발되어야할 필요가 있다.

본 연구에서 개발된 기술은

국정원, 국가 안전국(NSA): 테러, 국가 기밀 자료 유출 감시 및 국가 보안용도
검찰 및 경찰: 범죄집단의 정보 교환 및 인터넷 폰 통화 내용 감청
국세청: 탈세 및 돈 세탁 추적과 증거 해독
기업 및 공기관: 내부 기밀 정보의 유출 감시등의

국가 법률집행기관이나 국가적인 경쟁력을 갖는 국가기관이 법집행과 상업적인 기술 우위를 유지하기 위해 사용해야 하는 공익 기술이다.

가. 암호기술의 확산 및 일반화

국가가 독점하고 있던 암호화 기술이 일반으로 확산되면서 개인과 기업의 자료를 외부의 침입이나 내부자의 기밀 자료 유출을 막기 위해 보다 강력한 기술이 요구되고 있다. 특히 암호기술이 일반화되고 있는 미국에서는 테러 단체나 범죄조직, 국제 마약 거래상 등에 의해 암호기술이 악용될 상황을 심각히 우려하고 미 정부가 이에 대한 대응 마련에 고심하고 있다.

암호기술은 양날을 가진 칼과 같이 개인과 산업계의 프라이버시를 증대시키는 반면 범죄자와 테러리스트들의 불법행위의 방패막이가 되기도 한다(1993년 미국 Clipper Act Proposal; 1994년 백악관 보도자료; 1995년 9월 의회의 OTA). 이러한 이유로 국가 법 집행력의 보장, 국가 안보 등을 확보하기 위해서는 암호사용을 규제해야 한다(1997년 OECD암호정책 지침)는 주장이 각 국 정부에 의해 강하게 대두되고 있음. 특히 미국의 Key Recovery 정책은 법집행기관의 이 같은 고민을 해결하기 위해 대두된 궁여지책으로서 현재도 논란이 되고 있다.

나. 네트워크를 이용한 범죄 행위에 대한 국가적 대응 방안의 필요성

2001년 9월 11일 미국의 테러 사건 이후 Key Escrow정책이 새로운 조명을 받고 있다. 미국에서 발간되고 있는 Computer Weekly (2001. 10. 14)자에 따르면 테러 사건으로 미국 정부는 Key Escrow 정책의 도입을 다시 검토하기 시작하였으면 상당한 호응을 받고 있는 것으로 보도하고 있다. 전 내무부 장관(former home secretary)이자 현재 외무장관(foreign secretary) Jack Straw과 현재의 내무장관인 David Blunkett는 테러사건 이후 key escrow 정책의 도입을 토론하였다고 보도하고 있다.

미국 클린튼 행정부가 1993년 Clipper Act이후 1995년 12월 발표된 Clipper-II(소프트웨어 키 위탁 암호의 수출 기준안), 1996년 6월 Clipper-III (범세계적 정보 인프라에서의 프라이버시, 전자상거래, 정보보안, 공공의 안녕 촉진), 1997년 3월 Clipper-IV (1997년 전자 데이터 보호 법안)를 통해서 공공의 이익을 위한 개인 암호화키를 정부기관이 접근할 수 있도록 하려는 정책이 추진되어 오고 있다. Clipper-IV에서는 키 복구에 대한 규정과 키 인증기관, 키 복구 대행자, 키 복구절차, 책임과 사용 등에 관해 구체적으로 제시하고 있다. 이처럼 미국 정부가 key escrow 정책 수립과 인정에 적극적으로 나서고 있는 것은 향후 전자 상거래가 활성화되면서 암호화 기술이 발달, 정부의 법집행 능력이 크게 떨어질 것을 우려하기 때문이다.

현재까지 추진되어 온 key escrow 정책은 clipper chip과 capstone chip에서 시행이 추진되고 있다. Clipper Chip이란 개인이 사용하는 전화의 암호화키를 두 개의 다른 정부기관이 반쪽씩 보관하고 있다가 범죄 수사 목적이나 기타 중요한 목적으로 법원의 허락을 받는 경우 두 개 기관의 반쪽 키를 합하여 통신을 해독할 수 있도록 하려는 기술이다. Clipper Act에서는 제작자가 안전하게 제작한 유일한 순번(unique serial) 암호화키를 반으로 쪼개어 상무성 소속인 NIST와 재무성 산하의 automated Systems Division에 나누어 보관하도록 하고 있다. 우선 통화가 시작되면 세션키를 만들어 낸 다음 법률 집행기관의 접근을 가능하게 하는 LEAF(Law Enforcement Access Fields) 만들어 교환하게된다. LEAF가 없이는 통화가 시작되지 않는다.

제 3자가 LEAF를 위장하여 도청하려는 것을 방지하기 위해서는 세션 키를 Cip자체가 가지고 있는 키를 가지고 암호화하고 유일한 순번 번호와 checksum을 붙여서 family key라고 부르는 법률 집행기관만이 가지고 있는 키로서 암호화한다. 만약 법률 집행기관이 전화를 도청한 후 이를 해독하려면 family key를 가지고 있는 특별한 복호화 프로세서를 사용하여야한다. 그 프로세서를 가지고 통화자의 칩에 들어있는 순번 번호(serial number)를 복호화하고 법률적인 허락을 법원으로부터 받으면 비로소 key escrow기관 양측에 요청하여 완벽한 한 쌍의 키를 가지고 해독을 할 수 있게 된다. 이 전화의 암호화에 사용되는 skipjack 알고리즘은 80비트 키를 사용하고 있으면서 DES보다는 안전성이 훨씬 더 높아진 것으로 알려 져있다. 미국 정부에서 전문가를 초빙하여 토론을 벌인 결과 (1)컴퓨터 프로세서의 Power 비용이 18개월마다 반감된다는 가정하에 무차별 공격(brute-force)에 의해 skipjack을 해독하는 비용은 오늘날 DES를 해독하는 비용과 같아지기까지 36년이 걸린다. 즉 30년에서 40년 사이에 해독될 가능성은 없다. (2)공격의 단기적인 방법을 통해서 해독되어질 확실한 위험은 없다. (3)SKIPJack

내부구조가 법시행과 국가 보안 목적을 위해 분리되어져야 한다고 하더라도 SkipJack의 공격에 대한 내구성은 알고리즘의 기밀에 의존하지는 않는다 라는 결론을 내리고 있다 [Stallings, 1995, P.325].

Fortezza Card (Capstone Chip)에서는 미국무성의 메시지 시스템의 보안으로 채택되어 현재 2백만명 이상이 사용될 PCM-CIA통신 카드에 들어가는 칩이다. Capstone Chip은 Clipper Chip과 마찬가지로 유일한 번호를 사용하여 LEAF를 만들어 낸다. Capstone Chip을 사용하여 의사 난수를 발생하여 메시지를 암호화하고 전자 서명을 첨부하는 기능을 가지고 있다.

이러한 공공기관에 의한 암호키의 분배와 감시는 조세권의 실현과 공공의 안녕과 질서유지가 당위성으로 논의된다.

[조세권의 실현] 탈세와 비밀 자금 세탁의 경우, 자료를 암호화하여 저장하거나 전송하게 되면 발견 체포하더라도 거래 내역을 읽어보거나 범죄사실 입증이 불가능해짐. 즉, 공공의 안녕, 국가 안보와 외교 정책상의 이익확보, 그리고 조세권 확보의 입장에서 적법한 자료 접근권이 필요하게 되었고 이를 위해 미국정부에서는 키 복구 정책을 추진하고 있다 [박성준, 1999]. 조세기관의 과세자료를 은닉하는 것은 이미 일반화된 조세 회피 방법이지만, 암호의 사용은 정보가 디지털화된 오늘날 과세자료를 거의 완벽하게 은닉할 가능성을 제공, 조세 회피가 우려됨. 특히 돈 세탁을 통한 조세회피는 이미 사회적인 쟁점으로 부각.

OECD는 1998년 2월 12일 금융행위 실무 작업반(Financial Action Task Force) 의 보고서에서 범죄자들이 사용하는 새로운 유형의 돈세탁을 공개한 바 있으며, 여기서 인터넷 카지노 등을 이용한 청단 방법을 포함. 이 보고서에 따르면 범죄자들이 전자 자금 이체를 이용하여 돈을 예치하는 경우 수사 당국은 예금자들에 대한 정보를 입수할 수 없다고 밝히고 있으며, 특히 범죄자들이 불법으로 입수한 현금을 인터넷 카지노 등에서 칩으로 교환하고 다시 칩을 카지노 계좌에서 발행하는 수표로 바꾸는 경우, 합법적인 도박으로 얻은 수표이기 때문에 돈세탁이 가능하며, 인터넷상에서 카지노는 신용카드를 이용하는 고객들의 신분을 완벽하게 보호해주는 기술을 사용함으로써 새로운 문제를 야기하고 있다고 밝히고 있다.

[공공의 안녕 유지] 정부는 공공의 안녕 유지를 위하여 법률에 의하여 적법한 압수 수색 및 감청을 행사할 수 있으나 암호 사용증대는 범죄활동의 내용을 암호화하여 은닉할 수 있는 가능성을 증대시킴으로써 적법한 국가의 치안 유지 능력이 저하될 수 있음. 특히 2001년 9월 11일에 발생한 뉴욕의 World Trade Center테러 사건이나 펜타곤 충돌 사건 등은 결국 다국적군을 동원한 전쟁과 탄저병 공포 확산 등으로 번지고 있다. 이중에서 주시할 만한 사건은 이 사건을 주도한 것으로 지목되고 있는 오사마 빈 라덴이 전세계에 흩어져 있는 테러리스트들에게 포르노 사이트를 이용하여 포르노 사진속에 메시지를 숨겨서 전송하는 기술을 사용하였다고 보고되고 있다. 이는 최근 개발된 워터마크 기술을 사용한 것으로 현재로서는 해독이 불가능한 기술이다.

이처럼 범죄 집단들이 독자적으로 개발한 암호 기술을 이용하여 데이터를 전송하거나 저장

하는 경우 이를 추적하거나 해독할 길이 없다. 그리고 미국의 FBI, CIA, NSA 등에서는 범죄집단의 감시에 도청을 많이 사용하고 있으나 디지털 기술의 발달로 암호화된 인터넷 폰 기술을 사용하는 경우 이를 해독할 방법이 전혀 없다. 즉, 인터넷 폰과 같은 소프트웨어 중심의 통화 수단을 사용한다면 기존의 도청 기술만을 이용하는 정부 기관이 이를 감시할 방법은 더 이상 존재하지 않게 된다. 인터넷 폰은 전화를 통해서 전달된 음성이 디지털 형태의 파일로 바뀌어 전송되기 때문에 암호화가 가능하다. 만약 독자적인 소프트웨어를 개발하여 인터넷 폰 통화내용을 암호화하고 이를 상대방 컴퓨터로 보내어 음성으로 변환하는 기술을 사용한다면 정부기관이 범죄자들의 통화를 감시할 방법이 없다. 최근 암호화 기술이나 인터넷 폰 등은 대학원 학생이나 심지어 학부 학생들도 개발할 수가 있기 때문에 범죄조직에 의한 독자적인 기술 개발의 가능성은 더욱 높아져 가고 있음.

[국가 안보 및 외교정책상의 이익 보호] 국가 안보 및 외교 등에 관한 정보를 수집하는 정보활동은 냉전이후 이미 각국에서 일반화한 첩보활동이며 이러한 첩보활동의 수행에 있어 정보를 암호화하여 저장, 전송하는 경우 국가의 법집행력 확보가 곤란해짐. 특히 국제적인 테러활동이나 마약상 등에 의한 암호 사용은 국가 안보의 유지를 위한 법 시행력 확보에 악영향을 미칠 수 있음.

다. 암호기술 일반화와 키 복구 시스템의 대두

키 복구(Key Recovery)란 기밀성 목적의 암호화된 메세지를 해독할 수 있는 키 또는 관련 자료를 제3자에게 위탁하고 일정한 조건하에 위탁된 키와 관련자료, 또는 평문을 적법한 권한이 있는 자에게 인도하는 시스템이다. 현재 미국 정부가 추진 중에 있는 중인 키 복구시스템(Key Recovery System)은 국가 기관의 적법한 자료 접근 허용이 핵심이다. 이는 암호화 기술의 발전으로 정부의 법 집행력이 심각한 위협을 받게 되었기 때문임. 예를 들어 마약 거래, 유아 포르노, 국제 테러의 경우 데이터를 암호화하여 전송하는 경우와 음성통화를 디지털화하고 이를 암호화하여 전송하는 경우, 불법행위를 감시할 수가 없어지기 때문이다.

1998년 2월 연방 키 관리 기반(Federal Key Management Infrastructure)을 위한 FIPS(Federal Information Processing Standard)개발을 위한 연방 기술 자문위원회에서 발표한 키 복구 시스템은 "지난 20년 동안 암호의 사용은 꾸준히 증가해왔고 정부의 암호 사용에 대한 의존도는 컴퓨터 분야가 인터넷에 의존해 가는 것처럼 매우 급속히 높아져 왔다. 이러한 암호 사용에 대한 의존은 키의 분실/손상이나 암호 메커니즘 자체가" "법적인 권리를 가지는 사람이 접근하는 것을 막도록 오용되는 경우처럼" 국가기관의 암호키 접근을 가능하도록 키 복구 시스템을 제안한다고 밝히고 있다.

물론 미국의 키 복구 정책은 다음의 세 가지 시나리오에 의해 만들어 졌다. (1)어떤 단체(또눈 개인)을 위해 저장된 데이터의 복구, 불의의 사고에 의한 키의 유실 (2)어떤 단체를 위한 통신 데이터/저장 데이터의 복구, 혹은 감시나 감사 활동을 위한 목적 (3)법적으로 허가된 기관에 의한 통신 데이터/저장과 복구라고 그 목적을 밝히고 있다. 이처럼 여러 가지 목적을

위해 개인이나 단체가 사용하는 키를 key Recovery Agent에 맡겨두도록 되어 있지만 결론적으로 미국이 추진하고 있는 것은 범죄집단에 의한 암호문을 해독하기 위해 암호 key를 정부기관에 등록하도록 하려는 것이 중요 골자이다.

라. 비표준화 암호 기술 사용시의 문제점

키 복구시스템이 일반화된다고 가정하여도 비 표준화된 암호화 알고리즘이 범죄집단과 테러집단 등에 의해 사용되는 것이 가장 심각한 문제로 대두될 것임. 특히 암호 기술이 특정 두 집단간의 의사소통을 목표로 개발된다면, 그리고 이를 사용하는 집단의 수가 적을수록 기밀성은 높아짐. 따라서 범죄 집단이나 테러집단이 이러한 비 표준화된 독자적인 암호화 메커니즘을 사용할 가능성이 대단히 높다.

현재 국제적으로 수많은 비표준화 된 암호화 알고리즘과 아이디어들이 제안, 개발되고 있으나 표준화된 알고리즘으로 인정받은 것은 극소수이다. 표준화되지 못한 이러한 알고리즘들 중의 하나를 어떤 범죄 집단이 독자적으로 사용한다면 이를 해독할 수 있는 길은 거의 없다. 현재 각 알고리즘의 해독 가능성에 대해 말하는 것은 사실상 알고리즘이 공개되어 있는 상태에서 키의 길이에 따르는 난이도를 말하는 것이다.

알고리즘이 공개되어 있지 않은 경우, 이를 해독하기란 사실상 불가능할 것으로 생각되며 이를 위해 암호화된 메시지의 특성의 파악, 그리고 암호화된 메시지의 해독 가능성에 대한 연구와 실제 해독 작업을 추진하여야 한다.

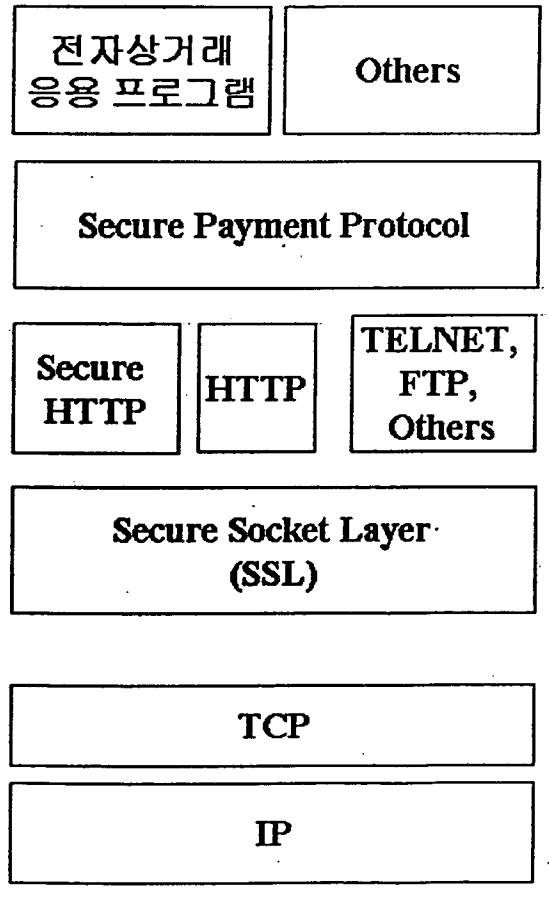
2. 기술 현황

암호화 알고리즘은 기본적으로 평문(Plaintext)과 키(Key)를 섞어 전혀 의미가 없는 문장으로 만드는 랜덤화(Randomizing)를 목표로 한다. 따라서 암호 알고리즘은 공개되어 있지만 키 값을 모르면서 원래의 평문을 해독하는 것은 불가능한 것으로 인식되어 왔다. DES 알고리즘의 경우 3회전, 혹은 6회전의 경우 본래의 평문을 해독하는 차분 암호공격(Differential Crypto-analysis) 기술이 제안되었지만 DES는 이러한 공격에 대해 안전하도록 16회전을 하고 있다. 1970년대 DES를 개발하던 개발자들은 이미 이러한 차분 공격에 대해 알고 있었다고 함. 즉, 키 값을 모르고서 암호문을 해독하는 것이 수학적으로 불가능한 것으로 인식됨. 현재까지 여러가지 시도가 이루어졌지만 모든 방법을 동원하여 key값을 찾아내는 시도만 있었을 뿐 암호화 알고림의 수학적 특성을 이용하여 암호문을 해독하거나 특성을 찾아내는 시도는 공개적으로 이루어진 적이 없다.

키크기	가능한 키의 수	m^2당 암호화	M^s당 10^6암호화
32비트	$2^{32}=4.3 * 10^9$	2^{31} ms =35.8 분	2.15ms
56비트	$2^{56}=7.2 * 10^{16}$	2^{55} ms =1142 분	10.01h
128비트	$2^{128}=3.4 * 10^{38}$	2^{127} ms =5.4 * 10^{24} 년	5.4 * 10^{18} 년
26비트	26!=4.03 * 10^{26}	2 × 10^{26} ms =6.4 * 10^{12} 년	6.4 * 10^6 년
순열(permutation)			

[그림 1-1] 모든 암호 키 탐색 방식의 소요시간

현재 전자상거래 시스템에서 가장 광범위하게 사용되고 있는 RC4는 SSL(Secure Socket Layer)는 네스케이프에 채용되었기 때문에 세계적으로 광범위하게 사용되는 상업용 암호화 알고리즘이다.



[그림 1-2] Web보안 시스템의 계층적 구조

수출 가능한 RC4 40bit 알고리즘에 대한 무차별 공격(Brute-Force Attack)이 공개한 된 것은 인터넷의 CypherPunks 집단에 의해서이다(Hal Finney, 1995: http://www.portal.com/~hfinney/sslchallong.html). 물론 NSA(National Security Agency)에서는 RC4 40비트를 깰 수 있는 실험을 끝내고 있었지만 이를 공개하고 있지 않았을 뿐이다 [Goldberg, and Wagner,

2001]. 곧 이어 영국에서 Adam Back, David Byers, and Eric Young들은 여러 대의 워크스테이션에서 돌아가는 빈 프로세스를 이용하는 소프트웨어를 가지고 RC4 40비트 알고리즘의 키를 성공적으로 찾아내었다고 공개하였다 [Adam Back, 1995]. 동시에 프랑스의 Damien Doligez[1995]도 같은 소프트웨어를 사용하여 RC4 40Bit 알고리즘의 키를 성공적으로 해독하였다고 선언하였다. 곧 뒤이어 Piete Brooks, Adam Back, Andrew Roos, and Andy Brown [1995] 들은 여러대의 컴퓨터에서 사용하지 않는 Idle Cycles을 기증받아 인터넷상에서 분산된 탐색으로 31시간만에 40비트 키를 해독할 수 있는 소프트웨어를 개발하였다고 선언하였다. 이러한 실험은 결국 키 길이가 짧은 (Bit) 알고리즘의 경우 무차별 공격(Brute Force)으로 해독이 가능하다는 것을 보여주고 있다.

유럽의 무선통신 GSM은 A5 알고리즘을 사용하고 있다. GSM에서 사용되고 있는 암호화 알고리즘은 A5로서 1995년에야 암호 공격에 대한 논의가 공개적으로 다루어 졌다 [Ross Anderson, 1995]. 이 알고리즘은 그 동안 공개되지 않았기 때문에 암호분야 연구에서 별로 주목을 받지 못하였으나 1995년 비로소 논의가 시작되어 A4의 공격에는 2^{40}의 공격정도만이 필요할 것이라는 의견이 있었다 [Ross Anderson, 1995]. 최근 이에 대한 결론은 A5공격에는 40비트 공격보다는 좀 더 어렵고 64비트보다는 약간 쉬울 수 있다는 것이다 [Goldberg and Wagner, 2001].

DES공격에 관한 논문이나 연구는 많이 나와 있다. DES가 미국 정부의 공식적인 암호 알고리즘으로 지정된 지 얼마 지나지 않아 Whit Diffie and Martin Hellman [1977]은 customer-chip을 장착한 DES암호 해독용 컴퓨터 구조를 제시하고 2000만 달러를 들여서 개발하게되면 하루만에 DES키를 해독할 수 있다고 제의하였다. 이러한 견해에 대해 대단히 뜨거운 논쟁이 있었다. 어떤 견해에서는 암호 키를 시도하는 경우 실패와 실패 사이에 평균 시간이 아주 작기 때문에 Diffie and Hellman의 제안은 절대 성공하기 어렵다는 주장을 하고 있다 [Bruce Schneiner, 1994]. 이 논쟁이 잠잠해졌을 때가 되어 DES알고리즘은 컴퓨터의 성능개선에 의해 결국 1990년경에는 안전하지 못할 것이라는 결론에 도달하였다. Gilles Garon and Richard Outerbridge [1991]은 1995년이 되면 1백만 달러 짜리 고객이 설계한 DES 전용 컴퓨터를 사용하여 9일만에 키가 깨어질 수 있다는 주장을 제기하였다. 1992년 Eberle and Thacker, 1992]는 아주 빠른 DES칩을 사용하여 8일만에 키를 찾아낼 수 있다는 주장을 하고 있다. 다른 주장으로서는 DES전용 칩이 아닌 일반적인 칩을 사용해서도 100백만 달러만 투자하면 30일반에 키를 해독할 수 있는 컴퓨터 개발이 가능하다고 주장하고 있다 [Peter Wayner, 1993].

가장 주목받은 주장으로서는 Michael Winer [1993]가 고객이 설계하는 DES전용 칩을 사용하여 컴퓨터 설계에 50만 달러, 실제 컴퓨터 조립에 1백만 달러를 투자하면 3.5 시간만에 암호 키를 찾아 낼 수 있다고 제안을 한 것이다. 이러한 3.5시간이라는 주장이 현재까지는 DES 키를 찾기 위한 표준시간으로 거의 인식이 되고 있다. 1996년 상업용 암호키로써 RC4 와 DES키를 찾아내는데 걸리는 시간을 추론하기 위한 전문가 그룹이 만들어 졌다 [Matt Blaze, Whitfield Diffie, ..., 1996]. 보고서에서 이들 그룹은 RC4암호 키는 400달러 짜리 FPGA칩으로 5시간만에 키를 찾을 수 있다고 주장하고 있다. 이들의 주장은 키 탐색 한 번에 0.08Cent

가 소요된다는 소위 '8-cent encryption'설에 근거를 두고 있는데 키를 한 번 시험할 때마다 256번의 반복과 한 번 반복마다 4번의 메모리 접근이 필요하여 적어도 1024번의 작동이 필요하다는 점을 감안한다면 초당 적어도 3천만번의 반복이 가능하다는 이들의 주장은 상당히 과장되어 있다고 생각된다. 초당 Giga Hz가 가능한 컴퓨터에서는 사실상 가능하지만 당시의 주장으로서는 현실성이 희박한 것으로 Goldberg and Wagner[2001]은 주장하고 있다.

최근 관심을 끌고있는 DES 해독 기술은 여러 사람이 동시에 나누어 가능한 키 값을 시도하여 의미 있는 문장이 나타나면 그것으로 키 값을 찾아내는 방법임. 예를 들어 세자리 다이얼식 금고 자물쇠를 여는 경우, A, B, C 세 사람이 1-300, 301-600, 601-999로 나누어 모든 가능한 숫자를 시험, 이중에서 어느 숫자라도 구멍을 맞추어 문이 열리면 이것을 키 값으로 생각하는 방법이다. 이 방식은 사실상 DES키를 찾는데 사용되어 왔으며 일부 성공적으로 56비트 키를 24시간만에 찾아내기도 했다.

이 같은 무차별적인 공격 방법(Brute Force Method)으로 키 값을 찾아내는 경우가 있으나 시간적인 제약이나 동원되는 인원문제 등으로 실제적으로 암호문 해독에 유용하지 않은 방법이다.

본 연구에서 시도하고 있는 것은 카오스 분석을 통해 암호 알고리즘의 메커니즘을 이해하고 이를 바탕으로 가능한 몇 가지 현실적인 제약조건 (암호문 내용관련, 키 관련)을 가지고 암호문을 해독하는 일반적인 방법을 찾으려는 시도이다. 그 첫번째 단계로서 네트워크에서 포착되는 데이터 분석을 통해 사용하고 있는 암호 알고리즘의 패턴을 찾아내려는 것이다.

3. 연구의 목표

가. 연구개발의 최종목표

본 연구는 네트워크상에서 발생되는 각종 범죄 행위를 적발하기 위한 시스템의 개발로서 다음의 기능을 충족하는 시스템을 개발하고자 한다.

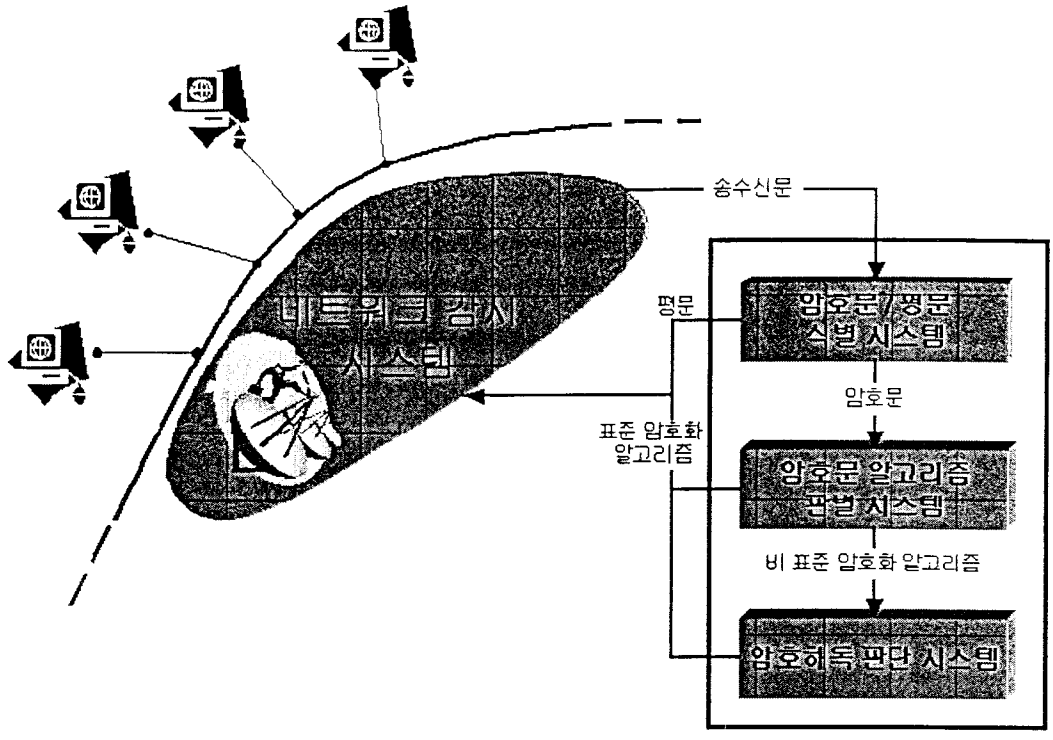
	최 종 목 표	세 부 사 항
암호패턴인식	암호문 / 평문 판별 95% 이상	o 암호문 : 암호 기술(DES, IDEA, MD5,...)을 적용한 Bit Stream, 혹은 Block 암호문 o 평문 : 암호화되지 않은 일반 문서(Plaintext), 혹은 에러 평문 o 판별기능 : 평문과 암호문, 에러 평문을 섞어 놓고 판별 결과 평가
암호패턴인식	암호 알고리즘 인식 기능 80%이상	o 암호 알고리즘 20개 이상 사용하여 암호문 생성 o 판별 기능 : 암호문을 대상으로 사용된 알고리즘을 구분하여 정확도 측정
네트워크감시	메시지 불법 송수신 5분 이내 판별	o 실시(Real-Time)성 판단 : 네트워크에서 내부자가 암호문을 전송하는 경우, 이를 판별하여 시스템 사용자에게 경고 메시지를 보내기까지 걸리는 시간을 5분이내로

<표 1-1> 연 구 목 표

o 본 연구의 최종 결과물은 암호문 판독 시스템과 네트워크 감시 시스템 두 가지이다.

[그림 1-3]에서 보는 바와 같이 네트워크 가시 시스템을 통해 의심스러운 사용자의 송수신문을 암호문 분석 시스템으로 전달하면, 암호문 분석 시스템은 송수신문이 먼저 암호문인가, 평문인가를 판단한다. 만약 송수신문이 암호문으로 판정 나는 경우, 어떤 암호화알고리즘을 사용했는가를 판단한다. 그런데 그 암호문이 비 공개된 자체 개발한 암호화알고리즘을 이용한 경우에는 해독 가능에 대해 판단을 하고, 이러한 분석결과를 네트워크감시 시스템으로 보낸다.

o 네트워크 감시 시스템은 지속적으로 네트워크를 감시하여 비정상적인 행위자. 예를 들어 중요 기밀문서를 유출하려는 내부자의 불법행위를 탐색하여 이를 네트워크 운영자에게 알려주는 시스템이다. 네트워크 감시를 위해서는 우선 암호문을 전송하는 사용자를 찾아낼 필요가 있다. 이는 허락 받지 않은 사용자가 문서를 암호화하여 보내는 경우 이를 일단은 비정상적인 송수신으로 간주하고 이를 다시 정말 분석하도록 시스템 운영자에게로 통보하여 준다. 그리고 암호화된 문서가 원래 어떤 형식의 문서인지 판단하고 이를 운영자에게 통보하는 기능도 필요하다. 만약 의심스러운 사용자의 행적(Behavior)이 정상적인 사용자의 범위를 넘어설 때는 즉시 1차적인 경고신호를 네트워크 관리자에게 보내고 일련의 행동이 위험수준을 넘어설 때는 사용자의 컴퓨터 자원접근을 봉쇄(LOCKING)하는 조치를 취하게 된다.



[그림 1-3] 네트워크상에서의 실시간 암호문 감시 시스템 구조

나. 연차별 연구목표

- 본 연구는 2년에 걸쳐 수행하며, 1차년도(1999년)에는 암호문 / 평문을 실시간으로 시스템을 개발하도록 한다.

구 분	연구개 발목표	연구개발내용 및 범위
1차 년도 (99년)	암호문/평문의 실시간 판별 시스템 개발	▶ 표준/비표준 암호화 알고리즘 프로그래밍(20가지 이상)
		▶ 통계적 특성분석을 통한 암호문/평문의 판별기술
		▶ 카오스 이론을 응용한 암호문/평문 판별 기술 개발
		▶ 암호문/평문 식별을 위한 임계치(thresholding) 설정
		▶ 암호문/평문 실시간 판별 및 리포팅 소프트웨어 개발
		▶ Data Mining 기법에 바탕을 둔 정상적인 사용자 Profile DB 설계
2차년도 (2000년)	암호문 알고리즘의 실시간 판별 시스템 개발	▶ 암호 알고리즘의 출력 패턴 분류기 및 DB 구축 ▶ 1차년도 암호화 알고리즘 보완 조사
		▶ 표준화/비표준화 암호 알고리즘의 식별 기술 개발
		▶ 신호처리 기술 및 카오스 분석기를 이용한 암호문의 해독 가능성 판별 기술 개발
		▶ 실시간 암호문 알고리즘 판별 소프트웨어 개발
		▶ 정상적인 사용자 Profile DB 구축
	네트워크 환경에서의 실시간 판별, 감시 시스템 개발	▶ Profiling 데이터와 사용자 행적의 비교 판단 기술 개발
		▶ 모의 시스템 및 실시간 리포팅 기술 개발
		▶ 시스템 통합 및 모의 실험을 통한 검증

<표 1-2> 연차별 연구목표

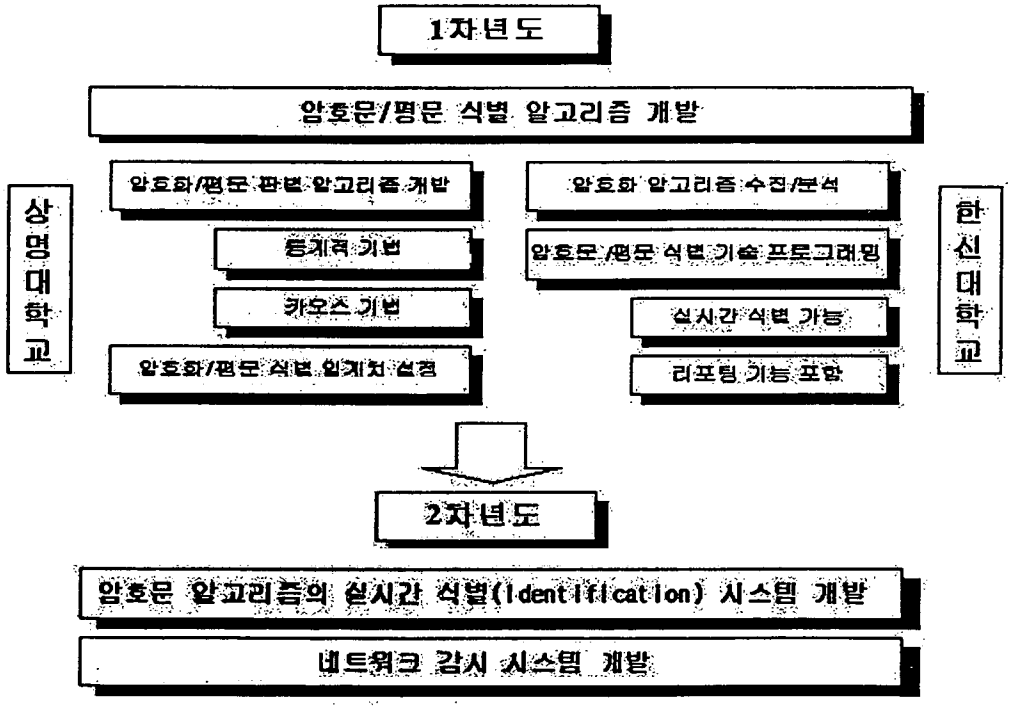
○ 1차년도(1999년)는 본 연구에서 활용하고자 하는 통계적 기법과 카오스(chaos theory) 이론을 적용하여 암호 알고리즘 패턴 인식 기술을 개발하고 , 이들 기술을 이용한 암호문 /평문 식별, 암호 알고리즘(3개) 패턴 인식 실시간 소프트웨어를 개발할 예정임. 네트워크 감시 시스템을 개발하기 위한 정상적인 사용자 Profile DB 설계와 실시간 네트워크 감시 리포트 모듈을 개발한다.

○ 2차년도(2000년)에는 암호화 알고리즘의 패턴 DB를 구축한다. 이는 각 암호화 알고리즘들이 평문과 키에 따라 각기 다른 출력을 나타내기 때문에 이들 특징을 수치화하여 DB를 구축하려고 함. 본 연구팀의 사전 연구에서 암호문의 출력값을 이진코드로 변환하여 분석하였더니 각 알고리즘의 출력 값에서 그 임의성(randomness)은 유사하지만 각 값

들이 시간에 따라 공간적으로 분포되는 현상은 차이가 있다는 것이 밝혀졌다. 따라서, 2차년도에는 카오스 분석기 (chaos analyzer)로 얻어진 패턴 정보를 공개된 암호 알고리즘에 적용하여 패턴 DB를 구축하고자 함.

o 또한, 비공개 암호문 알고리즘의 경우에는 암호화 알고리즘의 해독 가능성을 판단하기 위해서 신호처리 기술과 카오스 도구(툴)을 이용한 주기성 분석을 실시하도록 한다. 네트워크 감시 시스템 개발을 위해서는 사용자 Profile DB를 구축하고 이들 DB로부터 비정상적인 행위자의 구분하는 모듈을 개발한다.

o 2차 년도(2001년)후반에는 1, 2차 년도에서 개발된 시스템을 네트워크 상에서 적용하고 네트워크 감시 기술을 집중적으로 개발한다. 특히 Data Mining기법을 사용한 네트워크에서의 사용자 특징(Profiling) DB를 구축하고 협력 에이전트 기술을 사용한 분산 네트워크 감시 체계를 개발한다.



[그림 1-4] 연차별 연구목표

(1) 1999년(1차년도)

: 암호문 / 평문, 암호 알고리즘 패턴 실시간 식별 시스템 개발키 복구 시스템을 우회하여 (키등록 없이) 데이터를 전송하는 경우 이를 적발할 수 있는 기술을 개발하여야 한다. 전송되는 데이터가 평문인 경우, 숨겨지지 않기 때문에 문제가 되지 않는다. 법집행 당국이 관심을 가지는 것은,

(가) 키 복구시스템에 등록하지 않았지만 표준화된 암호 알고리즘을 사용하는 경우
(나) 키 복구시스템에 등록하지 않고 비 표준화된 독자적인 알고리즘을 쓰는 경우
(다) 평문을 가장한 암호문의 경우이다.

암호문 해독을 위해서는 가장 먼저 평문과 암호문을 구별할 수 있는 기술을 개발하여야 함. 본 연구에서는 최소한의 데이터를 가지고 평문/암호문 판별이 가능한 실시간 감시 기술을 개발하려고 함. 이를 위해서는 통계적 수치와 카오스 분석 값의 임계치를 적절하게 조정하는 알고리즘이 개발되어야 한다. 그리고 평문을 가장한 암호문의 식별 기술도 개발.

기초적인 암호 알고리즘 패턴인식 기술을 개발: DES, Seek, FEAL 등의 비밀키 암호화 알고리즘에 대한 패턴 인식 기술을 개발하려고 함. 특히 각 패턴을 네트워크 감시 시스템이 자동적으로 인식할 수 있도록 패턴을 수치화하는 작업이 필요.

네트워크 감시를 위한 각 사용자 특징 정보(Profiling) DB 설계: 네트워크 사용자들의 특징을 시간대, 터미널 위치, 주요 업무, 접근 DB 및 응용 프로그램, 허용 권한 등을 중심으로 DB화.

(2) 2000년(2차년도)

:네트워크 감시를 위한 내부 기밀 정보 유출 탐지 및 적발기술을 개발

1999년도에 개발한 사용자 Profile DB를 바탕으로 내부자 정보 유출을 감시 적발할 수 있는 확률 모델(Confirmative Model) 및 접근경로 추적(Access Path)을 위한 에이전트 기술을 개발.

본 연구에서 목표로 하고 있는 암호 판별 기술은 네트워크상에서 내부 데이터의 외부 전송을 실시간으로 모니터링(monitoring)하면서 수상한 흐름이 발견되는 경우, 구체적으로 이 문서의 흐름이 어떤 성격의 것인지 판별하여야 함.

내부자의 협의에 대한 최종적인 판단은 사용자의 class와 문서의 class, 접속 형태와 접속 시간, 그리고 타 문서의 취급 형태와 연계하여 판단하여야 하며 이는 사실상 외부자 침입 탐지(intruder detection) 시스템에서 사용하는 기술로서 사용자의 접근 경로 및 사용자 구분(Class), 접근 시간 등의 데이터와 정상적인 사용자의 Profile DB를 바탕으로 판별을 하게 됨. 1,2차 년도에서 개발된 기초적인 기술과 모델, 그리고 사용자 Profile DB를 마지막으로 통합하고 성능을 테스트하도록 한다.

제 2장 암호 알고리즘의 패턴 분석

1. 카오스 이론을 응용한 암호문 / 평문 판별기술

카오스 이론을 이용한 분석은 암호 알고리즘을 시스템에 상사(Mapping)시켜 평문과 key가 입력 신호가 되고 암호문이 출력 신호로 간주된다. 즉, 암호 알고리즘을 하나의 시스템이라고 할 때 시스템과 입력 값 혹은 시스템과 출력 값을 알면 나머지 하나의 값을 예측 혹은 추정할 수 있게 된다. 이러한 특성을 이용하여 암호문 알고리즘을 식별할 수 있는 시스템을 개발하고자 한다.

가. 카오스의 성질

기존의 통계학적 접근방법에서는 데이터의 특성을 연검정, 자기 상관계수, 회귀분석방법 등을 이용하여 규정하였으나, 고전적인 분류방식으로는 비선형 동태 방정식을 가지는 자료를 분류하기 힘든 단점을 가지고 있다. 이러한 시계열 데이터의 불규칙성은 혼돈이론(Chaotic Theory)을 통하여 설명되어 진다. 즉, 비선형 미분 방정식 또는 차분 방정식에서 랜덤한 요소가 전혀 없음에도 불구하고 제한되어지고 비주기적인 해가 존재한다는 사실이 밝혀진 것이다.

혼돈 이론은 간단한 모형계들을 사용하여 이들이 혼돈 운동을 할 수 있음을 보이고, 또한 이들 혼돈 운동에 대해 리아프노프(Lyapunov)발산 지수, 프랙탈 차원 등의 정량적 척도를 계산할 수 있다. 하지만 대부분의 실험적 측정에서 운동의 형태는 흔히 단일 변수의 시계열(time series)로 얻어진다. 이는 많은 경우에 우리가 계의 운동에 관련된 모든 변수들을 알 수 없기 때문이며, 또한 그것들의 존재를 안다 해도, 모든 변수들을 동시에 측정한다는 것은 대부분 불가능하기 때문이다.

카오스 특성(Chaotic)을 가진 자료의 경우에는 다음과 같은 특징을 가지고 있다.

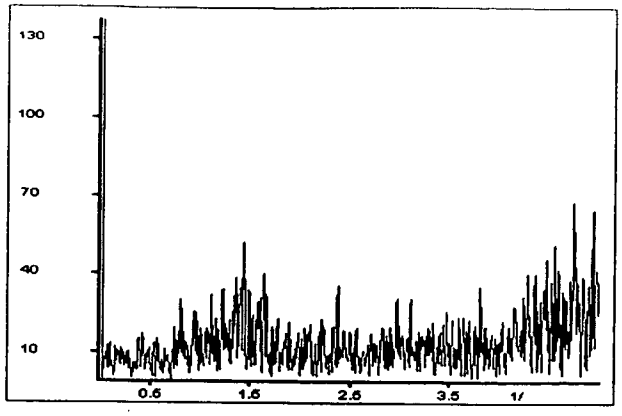
(1) 주파수가 넓은 영역에 분포한다. [이주장, 1993]

time-domain에 있는 데이터를 frequency-domain으로 변환하는 경우 넓은 주파수대역에서 피크(peak)가 발견된다. 다음의 그림은 혼돈의 특징을 가지고 있는 Henon map을 Fourier Transform을 수행한 후 주파수영역에서 나타낸 것이다.

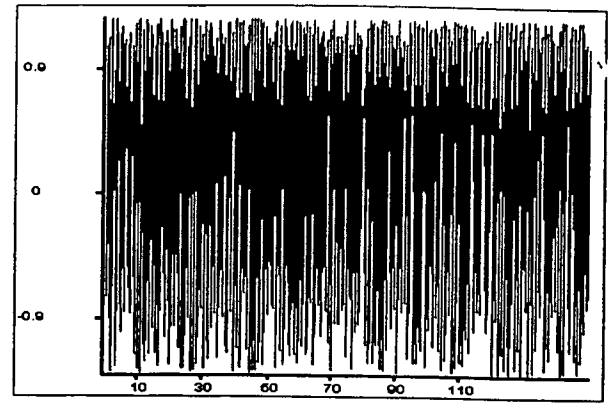
(2) 초기조건에 민감(Sensitive)하며 불확실성이 시간에 따라 증가한다.

위상공간에서 ΔX만큼 시차(時差)를 두는 두 초기조건에 대하여 시간에 따른 두 궤적간의 거리는 지수함수적으로 다음과 같이 증가한다.

$$d(t) = d_0 e^{lt} \cdot d_0 \quad \text{----------(식 2-1)}$$



[그림 2-1] Fast Fourier Transform을 수행한 후의 Amplitude Spectrum



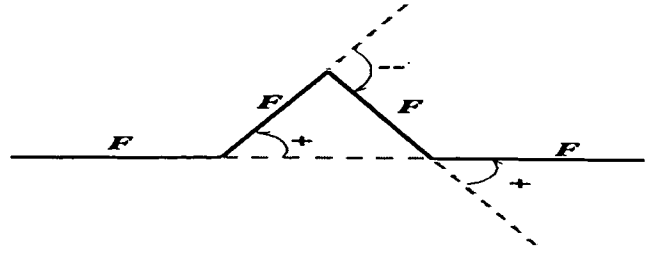
[그림 2-2] Henon Map Data Pattern

여기서 l을 Lyapunov exponent라고 하며 비선형 동태방정식에서의 외부변수에 의한 충격의 발산 정도 및 소멸시간을 나타낸다. 이와 같이 약간의 시차만을 가지고도 비선형함수는 다른 행태를 보여주는 초기민감성을 가지고 있다.

(3) Fractal 구조를 가진다.

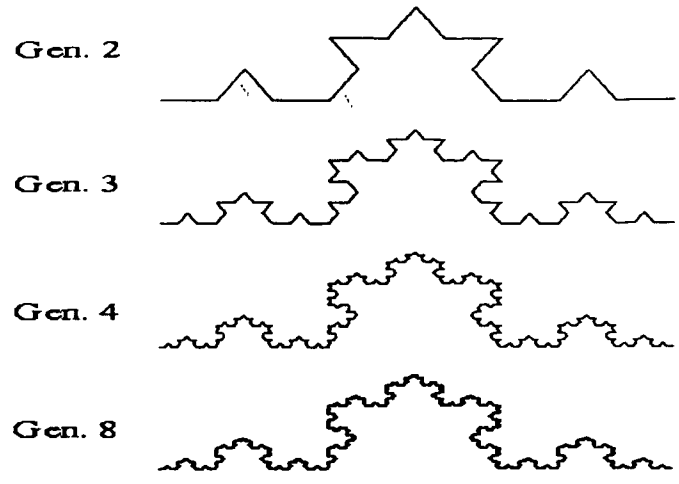
Chaotic한 특성을 가지고 있는 시스템의 경우에는 일부분을 잘라서 확대하면 다시 전체 모양이 재현되는 프랙탈구조, 즉 자기복제(self-similarity)의 특성을 가진다. 혼돈계가 프랙탈로 재현되는 이유는 특정계가 특정방향으로 발산하는 경우 주기적 또는 비주기적으로 발산방향과 반대로 굽어져 항상 유한한 영역에서만 자신의 구조를 반복해서 모양을 재구성한다.[프랙탈 영상압축, 1996]. 프랙탈 구조 또는 Koch snowflake로 알려진 Koch curve를 예로 들어 프랙탈 구조를 간단히 설명하면 다음과 같다[Turtle graphics and L-systems, 1996]

기하학적으로 4개의 단위길이(segment) F로 이루어지며 전체 길이에 대해 $\frac{1}{3}$인 지점과 $\frac{2}{3}$인 지점에서는 각각 양의 방향과 음의 방향으로 일정한 각도 $(+60^\circ, -120^\circ)$를 이룬다면 다음과 같은 그림이 형성된다.



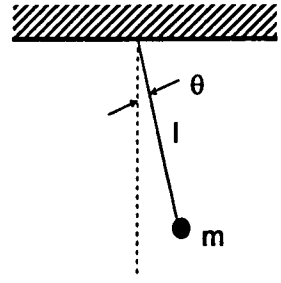
[그림 2-3] Koch 곡선의 생성

일차 생성된 Koch 곡선에 대하여 각 세그먼트에 똑같은 룰을 적용한다고 하면 세대가 진화함에 따라 계속 같은 모양이 반복되며 아주 복잡한 형태를 가지게 된다. 다음 그림은 세대가 반복된 후의 곡선의 그림을 나타낸다.



[그림 2-4] Koch curve generation

나. Attractor의 구성



[그림 2-5] 단진자 운동모형

질량이 m이고 길이가 l인 줄에 매달려 운동하는 단진자(Simple Harmonic Pendulum)의 경우를 가정해 보자. 운동에너지(kinetic energy)는

$$T = \frac{1}{2}mv^2 = \frac{1}{2}(l\dot{\theta})^2 \quad \text{----------(식 2-7)}$$

이며, 위치에너지(potential energy)는

$$V = -mgl\cos\theta \quad \text{----------(식 2-8)}$$

로 주어지게 된다. 여기에 Lagrangian을 적용하게 되면

$$L - T = \frac{1}{2}ml^2\theta'^2 + mgl\cos\theta \quad \text{----------(식 2-9)}$$

이며, Lagrangian eq. of motion은

$$\frac{d}{dt}(ml^2\theta') + mgl\sin\theta = 0 \quad \text{----------(식 2-10)}$$

또는

$$\theta'' = -\frac{g}{l}\sin\theta \quad \text{----------(식 2-11)}$$

로 주어지게 된다.[Boas, 1983] 이 방정식을 일반화시키면 다음과 같은 미분방정식으로 나타

낼 수 있다.

$$\ddot{x} + 2\zeta\dot{x} + x = 0 \quad \text{----------(식 2-12)}$$

위에서 감쇄상수 ζ = 0인 경우 주기함수의 해를 가지며 이것은 $x-x'$ 위상평면에 원으로 나타내진다. 이것을 제한궤도(limit cycle)이라 한다. 감쇄상수 ζ가 $0<\zeta<1$ 이면 위의 미분방정식은 감쇄진동해를 가지고 시간이 지남에 따라 $x=0$, $x'=0$으로 수렴하며 이는 $x-x'$ 위상평면상의 원점에 해당한다. 평형점이나 제한궤도와 같이 해의 궤적이 수렴하는 집합을 흡인집합 또는 끌개(attractor)라고 한다.

어떤 시간 간격 t 마다 X 값을 측정하여 얻은 시계열 데이터를 X(0), X(t), X(2t),... 등으로 표시하자. 전체 운동의 배후에 어떤 결정론적인 법칙이 있다면 어떤 시간에서의 X값은 과거의 X값들에 의존할 것이다. 얼마나 많은 X값들에 의존할 것인지는 원래 계가 갖고 있는 변수의 개수, 즉 기이한 끌개가 들어 있는 공간의 차원 값과 관계가 있다.[시계열 분석, 1996]

어떤 시계열 패턴(pattern)이 샘플링(sampling) 간격 δt로 n개의 데이터를 가지고 있다고 하면 시계열은 다음과 같이 표현된다.

$$\{X(t), X(t+\delta t), X(t+2\cdot\delta t), X(t+3\cdot\delta t), \cdots, X(t+(n-1)\cdot\delta t)\} \quad \text{----------(식 2-13)}$$

이때 시간지연(T)을 샘플링시간 δt의 2배로 잡고, 3차원 벡터열을 구하면 다음과 같다. (T = 2 δt)

$$\{X(t), X(t+2\cdot\delta t), X(t+4\cdot\delta t)\}, \{X(t+1\cdot\delta t), X(t+3\cdot\delta t), X(t+5\cdot\delta t)\}, \cdots,$$
$$\{X(t+(n-5)\cdot\delta t), X(t+(n-3)\cdot\delta t), X(t+(n-1)\cdot\delta t)\} \quad \text{-(식 2-14)}$$

이렇게 정해진 점들을 3차원공간에 찍으면 원래 계의 운동 성질을 보이는 3차원끌개를 얻게 된다. n값이 끌개의 원래 차원 값보다 같거나 큰 범위에서 잘 선택되면 이 벡터 열은 원래의 운동과 동일한 성질을 보이게 된다.

이와 같은 방법으로 재구성된 기이한 끌개는 물론 원래의 끌개와 정확히 같은 형태는 아니고, 일반적으로 변형된 형태를 갖는다. 하지만 발산(Lyapunov) 지수와 프랙탈 차원은 이러한 변형에 대해 변하지 않기 때문에 재구성된 끌개로부터 이들 값을 계산해 낼 수 있다.

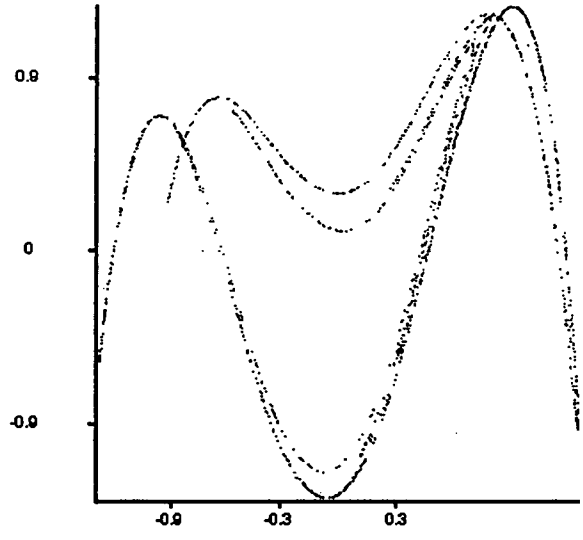
끌개의 재구성은 비선형 모델링, 잡음 축소, 비선형 예측 등 신호 처리의 예비 단계가 될 수 있다. 재구성된 끌개에 가장 적합한 운동 방정식을 찾음으로써 불규칙한 신호에 대한 비선형 모델링을 할 수 있다. 물론 끌개의 차원이 높아질수록 이 작업은 어려워지지만 선형 모델의 효용성을 훨씬 능가하는 모델링이 될 수 있다. 아래의 그림은 로렌쯔 방정식(Lorentz Eq.)을 가지고 끌개를 구성한 것이다. [그림 2-6]과 [그림 2-7]은 위 Henon Map을 이용하여 T를 1과 2로 주어본 경우이다.[Vandenhouten et. al., 1996]

다. Fractal Dimension

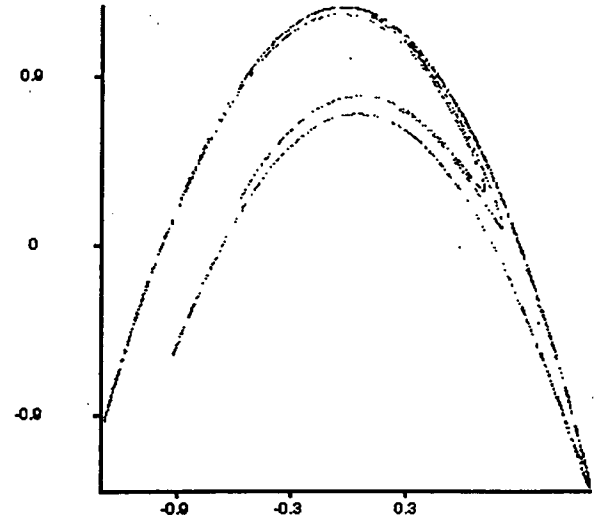
아날로그 신호를 $f(t)$라고 한다면 우리는 t를 적정한 간격 Δt로 샘플링하여 이산신호를 얻을수 있고, Δt를 충분히 작게 하면 원하는 신호를 복원할 수 있음이 수학적으로 증명되었다. 시간구간 [a, b]를 등분해서 샘플링한 신호 f를 N점의 샘플값 계열에서

$$f = (f_1, f_2, \cdots, f_N) \quad \text{----------(식 2-15)}$$

이라고 쓰면 f 는 N차원 벡터(vector)로 표현되게 된다.



[그림 2-6] Henon Map (tau = 2)

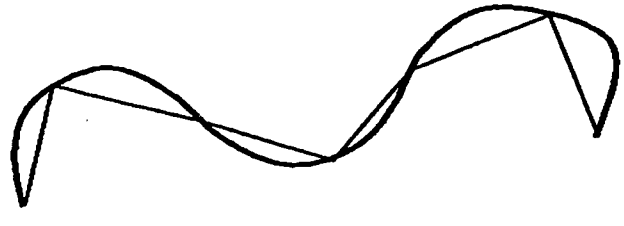


[그림 2-7] Henon Map (tau = 1)

(x_1, y_1)부터 (x_2, y_2)까지의 직선을 생각해 보면, 두 점간의 거리는 피타고라스의 정리에 의해

$$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \quad \text{----------(식 2-16)}$$

로 계산되어 진다. 그러나 곡선에 대해서는 곡선의 부분을 조각(segment)으로 나뉘어 각 점들간의 직선의 거리를 구하는 방법(polygonal approximation)을 적용하여 구하는 방법밖에 없다. 다음의 그림은 polygonal approximation의 방법을 나타낸 것이다.



[그림 2-8] Polygonal approximation of a curve

그러나 위와 같은 방법을 이용해서는 프랙탈 구조를 가지는 Koch snowflake의 길이를 측정하는 것은 거의 불가능하다. Mandelbrot는 1967년에 영국해안의 길이가 얼마인지 궁금하여 인접한 여러 국가의 백과사전을 통하여 조사해보는 과정에서 해안선이 복잡하고 작은 나라의 국경선이 더 길게 측정되고 있다는 사실을 알게되었다. Mandelbrot는 해안선의 길이를 측정한 측량자의 기본길이가 작을수록 길이가 더 길게 나온다는 재미있는 현상을 발견하였다.

위상 차원(topological dimension)과 마찬가지로 프랙탈 차원도 주어진 체적 $B(x_0, r)$의 주어진 집합 S를 둘러쌀 수 있는 C로부터 시작한다.[node 37, 1996] 직경이 r인 체적요소를 둘러싸기 위한 최소단위를 ε을 만족하기 위해서는 $r<\varepsilon$ 이어야 한다. 만약 S가 고정되어 있고 체적요소를 둘러싸기 위한 최소의 값을 C라 하면 최소의 체적요소의 개수는 $N(\varepsilon)$이어야 한다. Koch curve의 경우를 생각하면 $\varepsilon=\frac{1}{3^n}$으로 주어지며 $N(\varepsilon)=4^n$이다. ε 과 $N(\varepsilon)$간의 관계를 알기 위해 먼저 로그 값을 적용하면

$$\log\varepsilon = \log(3^{-n}) = -n\log 3 \quad \text{----------(식 2-17)}$$

$$\log N(\varepsilon) = n\log 4 \quad \text{----------(식 2-18)}$$

로 주어진다. ε에 독립적인 두 식의 비를 구하여 보면

$$-\frac{\log N(\varepsilon)}{\log\varepsilon} = \frac{\log 4}{\log 3} \fallingdotseq 1.26 \quad \text{----------(식 2-19)}$$

이라는 값이 나오며 이를 Koch 곡선의 프랙탈 차원이라고 한다. 즉, 프랙탈 차원 (D_f)는

$$D_f = \lim_{\varepsilon\to\infty}\frac{\ln N(\varepsilon)}{\ln 1/\varepsilon} \quad \text{----------(식 2-20)}$$

으로 정의되어 진다.[Ott et. al., 1994] 프랙탈 차원이 존재한다는 것은 끌개를 구성할 수 있다는 것을 의미하며 시계열 예측의 가능성과 예측범위를 판단할 수 있다.[백웅기, 1996]

라. R/S 분석과 Hurst Exponent

시계열 데이터의 경우에는 일반적으로 임의보행과정(Random Walk)을 따른다고 이야기되고 있다. 이런 임의보행과정의 시작은 아인쉬타인(Einstein)의 1908년에 브라운 운동에 대한 기술로부터 시작되었다.

$$R = T^{0.50} \quad \text{----------(식 2-21)}$$
$$\textit{where, } R = \textit{distanceecovered, } T = \textit{a timeindex}$$

우리는 이 공식을 경제학의 표준편차에 적용할 수 있다. 만약

$$x = x_1, x_2, x_3, \cdots, x_n \ (n \textit{ consecutive values}) \quad \text{----------(식 2-22)}$$

과 같은 n개의 연속적인 값들을 가지는 벡터라고 가정하면, 우리는 평균값과 표준편차를 다음과 같이 정의한다.

평균값 : $X_m = \frac{x_1 + \cdots + x_n}{n}$ ----------(식 2-23)

표준편차 : $S_n = n^{-\frac{1}{2}} \cdot \sqrt{(X_r - X_m)^2}$ ----------(식 2-24)

이다. 이를 정규화 시키기 위하여 Z-Score로 변환시키면 평균값은 0이 되며

$Z_r = (X_r - X_m)$ *where*, $r = 1, \cdots, n$ ----------(식 2-25)

으로 변환된다. 누적된 시계열(cumulative time series) Y를 정의하면,

$Y_1 = (Z_1 + Z_r)$ *where*, $r = 2, \cdots, n$ ----------(식 2-26)

이다. 여기서 가장 마지막 누적 시계열은 Z의 값이 항상 0이므로, 항상 0이다. 여기서 우리는 다음과 같이 시계열의 부합범위(adjusted range) R_n을 정의할 수 있다[Peters, 1994]. 즉, R_n 은 시계열 지수(time index) n인 시스템의 이동범위를 의미한다. 만약 $n = T$이라면, 우리는 초기 브라운 운동을 고려할 때, n의 증가에 따라 시계열 벡터 X가 독립적이라고 말할 수 있다.

$R_n = Max(Y_1, \cdots, Y_n) - \min(Y_1, \cdots, Y_n)$ *where*, $R_n > 0$ ----------(식 2-27)

이러한 개념을 브라운 운동이 아닌 시계열에 적용하는 경우 식 [2-1]은 아래와 같이 일반화 된다.

$(R/S)_n = C \cdot n^H$
where, *H is Hurst exponent*
R/S is ReScaled range ----------(식 2-28)

재구성된 범위는 평균값이 0이며, 국부적 표준편차를 의미하며 양변에 log를 취하면 다음과 같이 된다.

$\log(R/S_n) = H\log(n) + \log(c)$ ----------(식 2-29)

허스트 계수(H)는 상관 정도(correlation measure)에 영향을 미친다[Peters, 1991]. 상관정도는 다음과 같이 정의되어 진다.

$C = 2^{(2H-1)} - 1$
where, C = *correlation measure*
H = *Hurst exponent* ----------(식 2-30)

허스트 계수는 다음과 같이 3가지로 분류되어 질 수 있다.

(1) $H = 0.5$

무작위 보행(random walk)을 한다고 말한다. 즉, 발생되는 사건은 (식 36)에서 보면 알 수 C=0으로 아무런 상관관계도 가지지 않는다. 다시 말하면 현재의 추세는 미래에 아무런 영향을 미치지 못하는 것을 의미한다. 맨델로프(Mandelbrot)에 의하면 H=0.5인 경우에는 이차원 프랙탈을 가진다고 말한다. 이는 프랙탈 차원이 허스트 지수의 역수와 같음을 의미한다.

2) $0 \leq H < 0.5$

비지속성(antipersistent)을 가진다고 말한다. 만약 시스템이 현재의 주기에서 증가하는 경향을 가진다면 다음 주기에서는 반대로 감소하는 영향을 가진다. 이러한 비지속성은 H가 0에 가까워질수록 강하게 나타나며 C 는 -0.5인 음의 상관관계(negative correlation)를 가지게 된다.

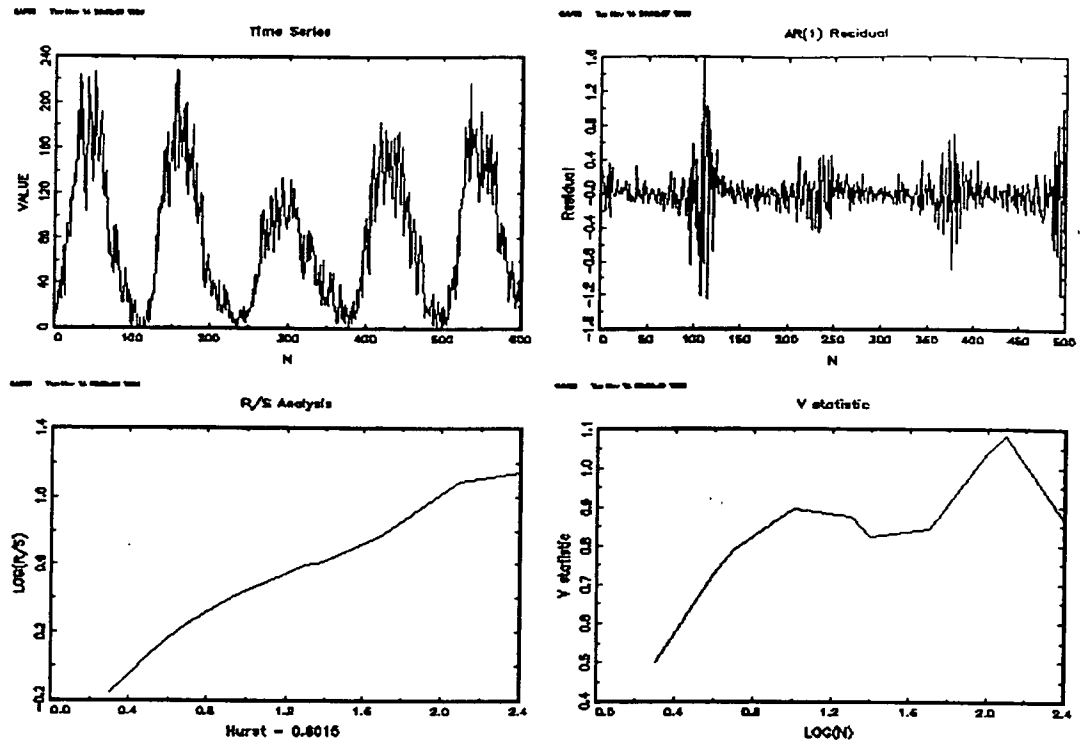
3) $0.5 < H \leq 1$

지속성(persistent) 또는 강제적 추이(trend-reinforcing)를 가진다고 말한다. 만약 현재주기가 증가(감소)하고 있다면 다음 주기에서도 증가(감소)하는 경향을 가진다. H가 1에 수렴하면 C는 2에 수렴하게 된다. 만약 H=0.60이라면 마지막 움직임이 양의 방향으로 증가하였다면 다음 움직임도 0.6의 확률을 가지고 증가한다고 말할 수 있다.

카오스계로부터 생성된 시계열의 Hurst지수 H는 N의 값이 작다면 추세성을 보이기 때문에 $H>0.5$ 이지만 N이 커진다면 현재의 시계열 값이 미래를 예측하는데 도움을 주지 못하기 때문에 H가 0.5로 수렴되어 가는 과정을 볼 수 있다. 따라서 N이 증가함에 따라 H값이 변하는 시점의 N은 기억효과의 지속성을 의미한다고 볼 수 있다. 다음의 그림은 카오스계의 성질을 가지고 있다고 알려진 혹점의 생성개수를 이용하여 R/S분석 및 Hurst 지수를 측정한 것이다.

마. Lyapunov Exponent

혼돈 시스템의 특징중 하나는 "초기조건의 민감성"이다. 비선형 동태 방정식을 가지는 시스템처럼 랜덤한 요소를 가지고 있는 경우에는 어떤 시점에서 발생한 충격(shock)이 유한한 시간이 지난 후에 영향력을 모두 잃어버리게 된다. 즉, 시스템의 운동방향은 충격에 따라 어떠한 방향으로 진행되게 되고 소멸되어진다. 이런 시스템의 확장 또는 수축을 의미하는 것이 리아프노프 계수(Lyapunov exponent)이다. 그러나 일반적인 소산(dissipative)시스템에 있어서 추세성 및 변동성을 보는 방법은 단순히Fourier Spectrum을 보거나 시간에 따른 출력값을 본다고 해서 알아낼 수 없다. 이런 경우 위상공간(Phase space)에서의 고찰이 가장 일반적이며,



[그림 2-9] 흑점 생성 개수에 따른 R/S 분석 및 Hurst 지수

그 속에서 Fractal 구조를 발견할 수 있다면 리아프노프값을 측정할 수 있다.[이주장, 1993] k-dimensional벡터 x_n을 가정하여 보면

$$x_{n+1} = G(x_n) \quad \text{----------(식 2-31)}$$

과 같이 나타낼 수 있다. 여기서 원래의 오비트(orbit)와 멀어진 오비트를 고려하 면, δx_n을 무한벡터라 가정할 때 $x_{n+1} \rightarrow x_n + \delta x_n$ 로 나타내어지며

$$\delta x_{n+1} = DG(x_n)\delta x_n \quad \text{----------(식 2-32)}$$

*Where, DG(x) = k*k Jacobian Matrix of partial derivative of G(x)*

라 할 수 있다. 여기서

$$y_n = \frac{\delta x_n}{|\delta x_0|} \quad \text{----------(식 2-33)}$$
$$y_{n+1} = DG(x_n)y_n$$

이라고 정의할 수 있으며, y_n은 탄젠트 벡터 또는 탄젠트 스페이스라고 부른다. 위 수식에서 보듯이 y_n은 초기조건 $\{x_n\}$과 초기의 탄젠트벡터 y_0에 의존하여 변화된다.

여기서 주목할 것은 y값이 매회 반복됨에 따라 탄젠트 공간이 확장되어지는지 수축되어지는지를 알아보는 것이다. 여기서

$$h(x_0, y_0) = \lim_{n \to \infty} h(x_0, y_0, n) \quad \text{----------(식 2-34)}$$

또는

$$h(x_0, y_0, n) = \frac{1}{n} \ln|y_n| \quad \text{----------(식 2-35)}$$

으로 나타낸다. 여기서 $h(x_0, y_0)$를 Lyapunov exponent(리아프노프 계수)라고 부르며, $h(x_0, y_0, n)$을 유한시간(Finite time) 리아프노프 계수라고 한다.[Ott et. al., 1994].

Lyapunov 지수는 계의 자유도(Degree of Freedom)만큼 존재한다. 자유도 2를 갖는 계에 대해서 주 점의 시간경로는 서로 분리되거나 합해진다. 만약 분리점들이 항상 서로 접근하는 방향으로만 움직인다면 Lyapunov지수는 음의 값을 가지게 되며 양의 값은 서로 멀어지는 발산의 형태를 가진다. 이런 Lyapunov 지수의 특성에 의해 계는 축소 또는 확장되어지고 흡인집합에서 주름진 형태나 이상한 형태의 끌개를 구성하게 되는 것이다.

바. 상관 차원(Correlation dimension)

시계열 자료에 있어서 가장 좋은 예측모델은 운동방정식을 알고 있는 경우일 것이다. 그러나 비선형 동태 방정식의 형태를 가지는 시스템의 경우에는 운동방정식으로 나타내는 것에 한계가 있을 뿐만 아니라 원하는 방정식을 찾아내기도 어렵다. 일반적으로 예측모델의 구성에 있어서 우리는 가장 조심스럽게 선택하여야 하는 부분 중에 하나가 독립변수의 개수를 지정하는 일이다. ARMA나 ARIMA와 같은 고전 통계모델의 경우에는 자동 상관 계수(AutoCorrelation Factor : ACF) 등을 이용하여 모델의 p값과 q값을 찾아내고 있으나, 비선형 동태방정식의 경우에는 이러한 해를 찾기가 힘들다. 이런 경우에는 카오스 검정에서 가장 중요한 단계로 취급하는 매립과정과 프랙탈 디맨젼, 상관 차원을 이용하여 해결한다.

시계열 $\{a_t\}$로 m-차원에서 매립화 시키는 과정은 $(a_t, a_{t+1}, \cdots, a_{t+m-1})$과 같이 단순히 관측치 m개를 중첩시킴으로써 완료된다. 매립과정을 통해 우리는 시계열 자료가 결정론적 설명력을 가진다고 말할 수 있으며[Takens, 1996] 원래의 상태벡터 $\{x_t\}$대신 $\{a_t^m\}$을 살펴봄으로써 저차원 카오스적 끌개가 있는지를 검정할 수 있는 근거를 제시하였다.

Grassberger와 Proccacia에 의해 제시되어진 상관차원은 시계열 자체가 하나의 변수에 의해 관찰되어지므로 1에서 2사이의 값밖에 가지지 못하는 한계를 가지고 있다. 그러나 위상공간에서 시계열을 재구성하는 경우에는 모든 독립변수들을 관찰할 수 있으므로 시계열을 위상공간에 재구성하는 것이 일반적이다. 이에 근거하여 제시된 상관 차원은 매립차원 m=2부터 증가 시면서 프랙탈 구조의 반경(diameter)을 증가하여 측정한다.

$$C_m(R) = \left(\frac{1}{N^2}\right) * \sum_{i,j=1}^{N} Z(R - |x_{i-x_j}|) \quad \text{----------(식 2-36)}$$

Where, $Z(x) = 1$ *if* $R - |x_i - x_j| > 0$; 0 *otherwise*

N = *Number of observations*

R = *distance*

C_m = *correlation integral for dimension m*

여기서 $Z(x)$는 Heaviside Function이라고 불리우며, 상관차원은 t와 $t+1$점간에서 프랙탈 반경 안에 두 점이 있을 확률을 의미한다.

사. 매립이론

시간 영역의 분석은 Taken's의 매립 이론 (embedding theory)을 이용하여 1차원 시계열을 m차원 시계열로 바꾸고 d-차원공간에서 계의 규칙성을 phase diagram과 return map으로 살펴보았다.

Determination of the time-delay value

스칼라 시계열에서 m-차원 위상공간을 재구성하기 위해서는 시계열 데이터의 관계를 조사하여 상호 독립적인 좌표를 설정해야 한다. 시간지연 값은 데이터간의 AMI (Average Mutual Information)을 측정하여 결정한다.

시간 지연 값 결정 방법은 먼저 Chiper text를 식 (1)과 같이 벡터로 표현하고, τ(시간지연 값)을 1부터 순차적으로 증가시켜 가면서 반복적으로 AMI를 측정하도록 한다.

$$\mathbf{x} = (x_0, x_1, x_2, \Lambda, x_i, \Lambda, x_n)^T \quad \text{----------(식 2-37)}$$

이때, τ 값에 대해서 $\mathbf{x}$의 시간지연 벡터 $\tilde{\mathbf{x}}$를 식 (2)와 같이 구하여 $\mathbf{x}$와 $\tilde{\mathbf{x}}$간의 entropy를 식 (3)을 이용하여 측정한다.

$$\mathbf{x} = (x_{-\tau}, x_{1-\tau}, x_{2-\tau}, \Lambda, x_{i-\tau}, \Lambda, x_{n-\tau})^T \quad \text{----------(식 2-38)}$$

$$S = -\sum_{ij} p_{ij}(\tau) \ln \frac{p_{ij}(\tau)}{p_i p_j} \quad \text{----------(식 2-39)}$$

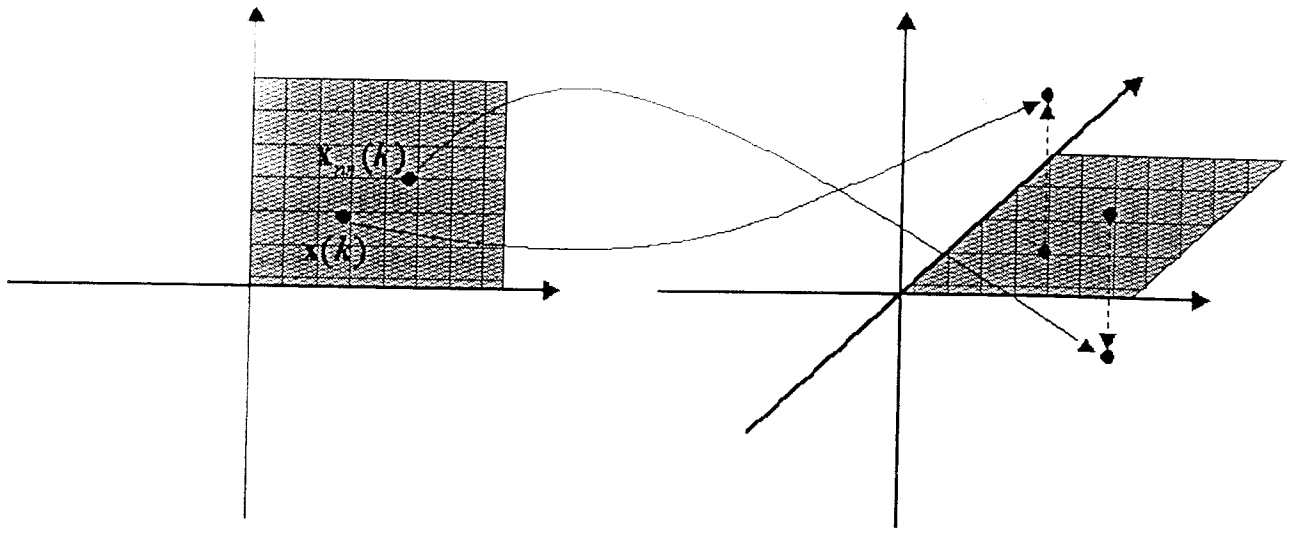
AMI가 첫 번째 극소값을 가질 때 벡터 $\mathbf{x}$와 $\tilde{\mathbf{x}}$는 서로 독립적인 좌표를 구성하게 되므로 S(entropy)값이 극소값을 갖는 τ를 시간지연 값으로 사용한다.

Determination of the minimum embedding dimension

AMI를 통해서 결정된 τ 값을 이용하여 1차원 벡터 $\mathbf{x}$의 차원을 하나씩 증가시켜 구성한다. 즉, 위상 공간을 m차원으로 구성하면 식(4)와 같다.

$$\mathbf{x}(k) = (x_k, x_{k+\tau}, \Lambda, x_{k+(m-1)\tau})$$

$$k = 1, 2, \Lambda, n-(m-1)\tau \quad \text{----------(식 2-40)}$$



[그림 2-10] FNN에서 위상공간의 변화에 따른 이웃의 위치 변화

식 (2-40)에서 m은 매립차원, τ는 시간 지연 값을 나타낸다. $\mathbf{x}(k)$에 가장 가까운 이웃 벡터를 찾아낸다. 이를 $\mathbf{x}_{nn}(k)$라고 하자. 만약 $\mathbf{x}_{nn}(k)$가 진실로 $\mathbf{x}(k)$의 가장 가까운 이웃이라면, 기하학적 사영에 의해서가 아니라 동역학적 이유에 의해서 이웃이 될 것이다. $\mathbf{x}_{nn}(k)$역으로 가 현재의 m의 크기가 위상구조를 완전히 펼칠수가 없어 사실 더 높은 차원의 위상 공간으로부터 투영에 의한 기하학적인 이유로 인해 $\mathbf{x}(k)$에 대해서 어떤 m의 값에서 이 모든 거짓 이웃이 완전히 없어지는 차원을 매립차원으로 결정한다. 이 방법은 Cao(1997), Abarbanel(1992), Kantz(1999)등에 의해서 발전되어 최근에는 신뢰성 있는 알고리즘으로 발전해 왔다.[3].

Kantz(1999년)의 FNN(false nearest neighbors)는 (식 2-41)과 같다.

$$\frac{|x_{i+(d-1)\tau+1} - x_{nn(i)+(d-1)\tau+1}|}{\|y_i - y_{nn(i)}\|} > Z \quad \text{----------(식 2-41)}$$

식 (2-41)에서 좌측변이 R보다 크면, 거짓 이웃이 되고 반대로 작게 되면 참 이웃이라고 판단할 수 있다.

Recurrence Plot & Return Map

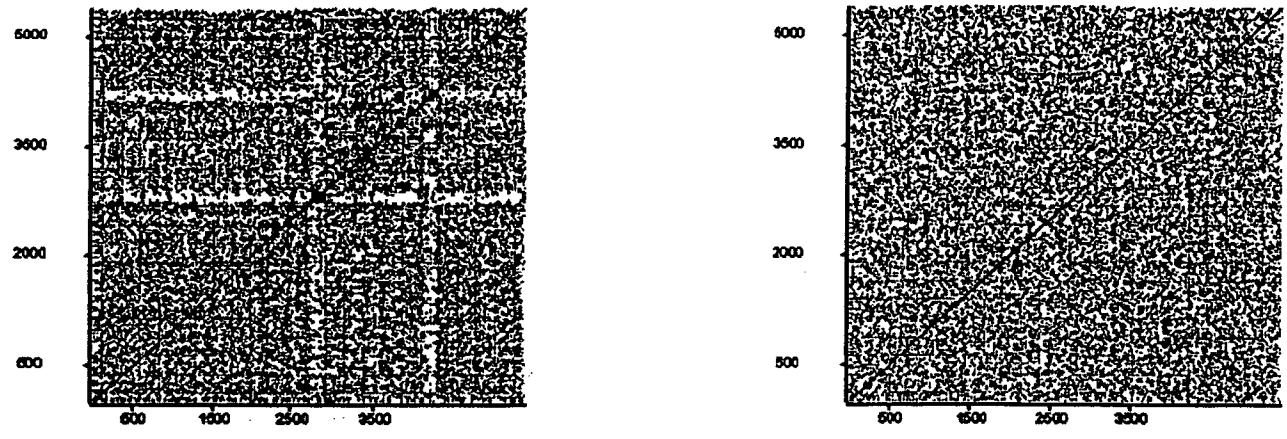
Recurrence plot는 식 (2-42)을 ε의 변화에 따라 이용하여 R_{kl}값이 1이면 좌표

$$R_{kl}(\varepsilon) = \Theta\left(\varepsilon - |\mathbf{x}(k) - \mathbf{x}(l)|\right) \quad \text{----------(식 2-42)}$$

$$\Theta(y) = 0, \quad y \le 0,$$

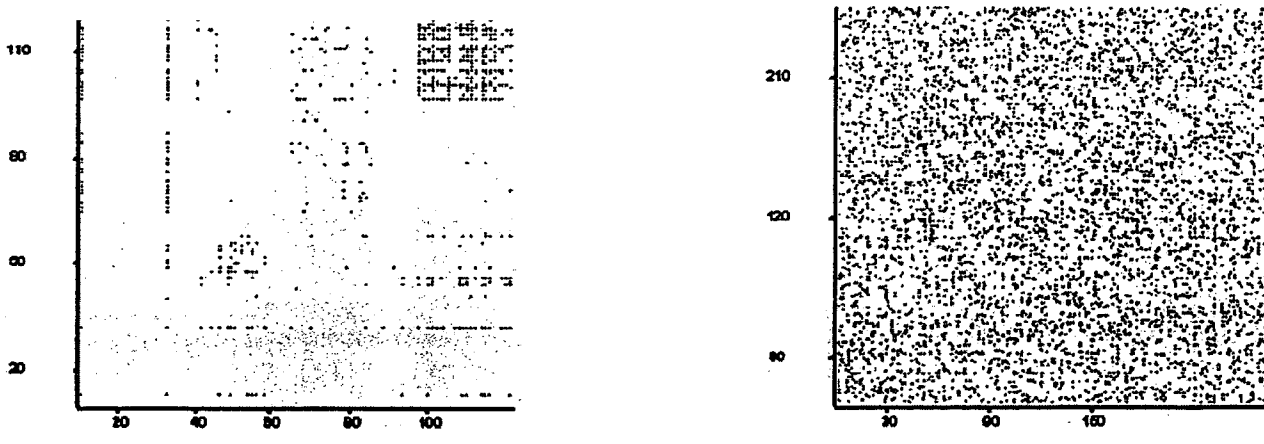
여기서, $\Theta(y)$는 Heaviside function이고, $\Theta(y)=1, \quad y>0$

Return map은 시스템에 내재되어 있는 가장 간단한 동력학적 구조를 관측하는 방법으로 시간지연 값을 이용하여 2차원 벡터를 각각 x, y좌표로 하여 2차원 평면에 점으로 구성한 가장 간단한 phase diagram이다.



(1) plaintext (2) ciphertext

[그림 2-11] 평문과 암호문 recurrence plot



(1) plaintext (2) ciphertext

[그림 2-12] 평문과 암호문의 return map

이처럼 평문과 암호문의 구별은 여러 가지 방법으로 가능해진다. 주파수 공간이나 Recurrence Plot, Return Map등을 통해 평문과 암호문이 확연한 차별성을 보이고 있다. 위의 분석 방법은 1000자 정도의 평문과 암호문을 가지고 분석한 결과이다.

본 연구진은 50자 - 100자 이내의 짧은 송수신문의 암호문/평문의 식별을 위하여 return map, recurrence plot의 계수값을 지표로 활용하여 식별실험을 한 결과 80%이상 구분이 가능하며, 300자 이내에서는 95%이상의 정확도를 갖는 것으로 판단된다.

카오스에 의한 암호문/평문의 임계치는 다음과 같다.

	Embedding dimension(Ed)	Return map correlation(AC)	Recurrence plot pattern clustering number(CM)
Critical criteria (임계치)	평문 : $0 < Ed \leq 7$ 암호문 : $7 < Ed < \infty$	평문 : $0.5 \leq AC < 1.0$ 암호문 : $0 < AC < 0.5$	평문 : $0 < CM < 5$ 암호문 : $CM > 6$

<표 2-1> 암호문 / 평문 식별을 위한 임계치

2. 암호문 알고리즘의 실시간 식별 시스템

암호알고리즘의 안전성 여부를 평가하기 위해서는 실제로 암호해독을 심도있게 실시하여 알아보는 방법 외에는 없다.

이에, 본 연구에서는 기존의 암호알고리즘이나 새로운 알고리즘의 암호해독 공격에 대한 안전성을 평가하는 방법으로 암호알고리즘을 시스템에 상사시켜 시스템의 입출력 신호를 분석하여 그 신호의 랜덤특성을 가지고 암호알고리즘을 평가하는 방법을 제안하고자 한다. 이를 위해서 Takens의 이론 [1]을 바탕으로 암호문의 시계열을 다차원 위상공간에 투영하여 내부적인 규칙성을 찾아내고, return map과 recurrence plot등 정성적인 기준으로 평가하고자 한다. 비록, 이 기법들은 매우 단순한 방법이지만 신호 또는 시계열에 내포되어 있는 계의 규칙성의 일면을 살펴보기에는 효율적이다.

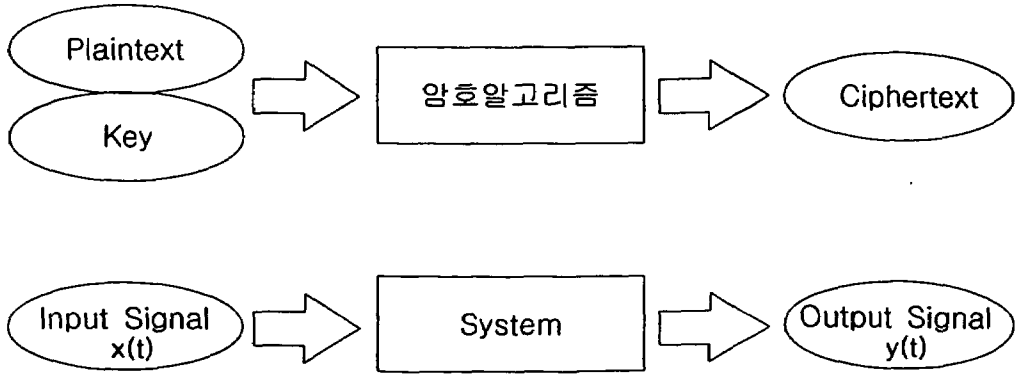
또, 주파수 영역 분석을 통해서 주기적인 특성을 관측하여 일정한 궤적이나 분포를 갖는 신호를 식별하도록 하였다.

3. 분석방법

본 연구에서는 암호문을 해석하는 방법으로 시간 영역 (time domain) 분석과 주파수 영역 (frequency domain) 신호 해석 기법을 이용하도록 한다.

가. 암호알고리즘과 시스템의 상사(equivalence)

암호알고리즘의 무작위 특성을 평가하기 위해서 시스템에 대한 접근으로 해결하고자 한다. <그림 1>과 같이 우선 암호알고리즘을 시스템에 대응시키면 시스템의 입력신호(input signal)은 평문(plaintext)과 키(key)가 되고 시스템의 출력신호(output signal)은 암호문(ciphertext)이 된다.



[그림 2-13] 암호알고리즘과 시스템의 상사

나. 시영역 신호 해석

시간 영역의 분석은 Takens의 매립 이론 (embedding theory)을 이용하여 1차원 시계열을 m차원 시계열로 바꾸고 d-차원공간에서 계의 규칙성을 phase diagram과 return map으로 살펴본다.

스칼라 시계열에서 d-차원 위상공간을 재구성하기 위해서는 시계열 데이터의 관계를 조사하여 상호 독립적인 좌표 설정과 시계열의 계의 규칙성을 관측할 수 있는 차원 (d)를 결정해야 한다.

수식 (2)의 평균 상호정보량 (AMI)은 시계열의 공간적 분포를 조사하여 상호 독립적인 좌표를 구성하는 시계열을 결정하는 방법이다. 시간지연 (delay time)은 AMI의 첫번째 극소값을 갖는 시점을 delay time로 선정하게 된다 [2].

$$s_n = \left(s_{n-(m-1)\tau}, s_{n-(m-2)\tau}, \Lambda\ , s_n\right)^T$$

$$S = -\sum_{ij} p_{ij}(\tau)\ln\frac{p_{ij}(\tau)}{p_i p_j} \quad \text{----------(식 2-43)}$$

시간 지연을 산출한 다음에 시계열 (신호)의 다이나믹한 계의 동태를 담을 수 있는 그릇의 크기를 결정하는 매립차원을 결정해야 한다.

매립차원의 결정은 끌개차원(d) 이하에서는 위상공간 궤적 상의 가장 가까운 이웃점들의 대부분이 거짓 이웃이 되나 끌개차원에 접근함에 따라 거짓 이웃의 수가 줄어드는 성질을 이용한다. 이 방법은 Cao (1997), Abarbanel (1992), Kantz (1999)등에 의해서 발전되어 최근에는 신뢰성 있는 알고리즘으로 발전해 왔다 [3].

$$\frac{\left|x_{i+(d-1)\tau+1} - x_{nn(i)+(d-1)\tau+1}\right|}{\left\|y_i - y_{nn(i)}\right\|} > Z \quad \text{----------(식 2-44)}$$

수식 (2)는 Kantz가 제안한 False Nearest Neighbor 방법으로 차원을 증가시켰을 때, 이전 차원에서 가장가까운 이웃이 여전히 문턱치 내에 있는 비율이 100%가 되는 차원을 매립차원으로 산정한다.

이상의 과정을 거쳐서 d-차원 벡터를 산출한 후에, Return map과 recurrence plot 등을 살펴보도록 한다.

Return map은 가장 간단하게 계에 내재되어 있는 동력학을 확인하는 방법으로서 유용하며, recurrence plot는 시계열에서 시간적으로 인접한 점들이 재구성된 위상공간에서도 근접거리를 유지하는 정도 및 주기적인 성질들을 나타내는 2차원 그림을 말한다.

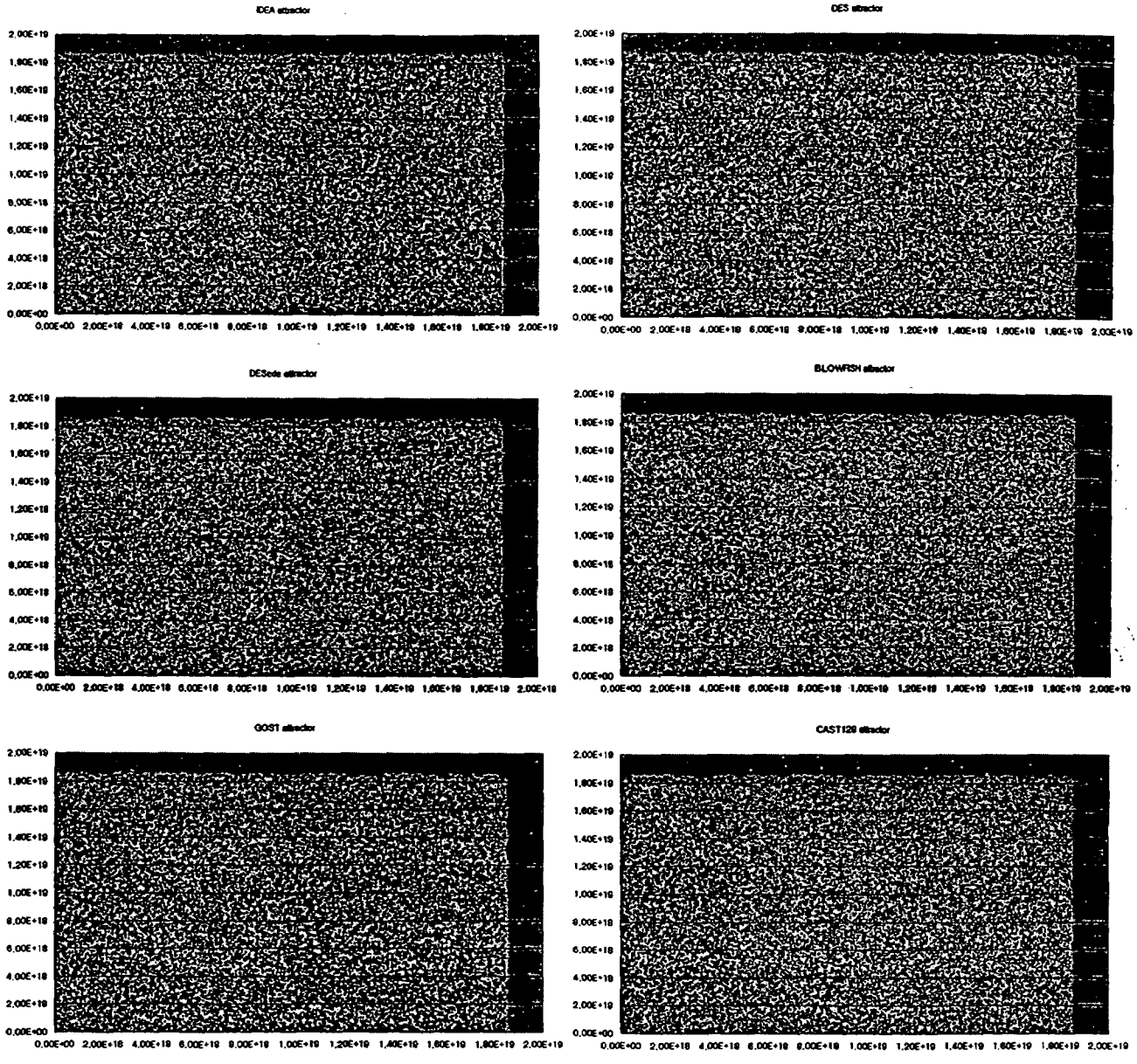
파 오 손 면

파 오 손 면

가. Return map

[그림 2-14], [그림 2-15]에서 확인한 바와 같이 시계열 데이터는 전형적인 임의의 값을 보여주고 있어 예측이 거의 불가능하다. 그러므로 다른 차원에서 접근해야 할 필요가 있다.

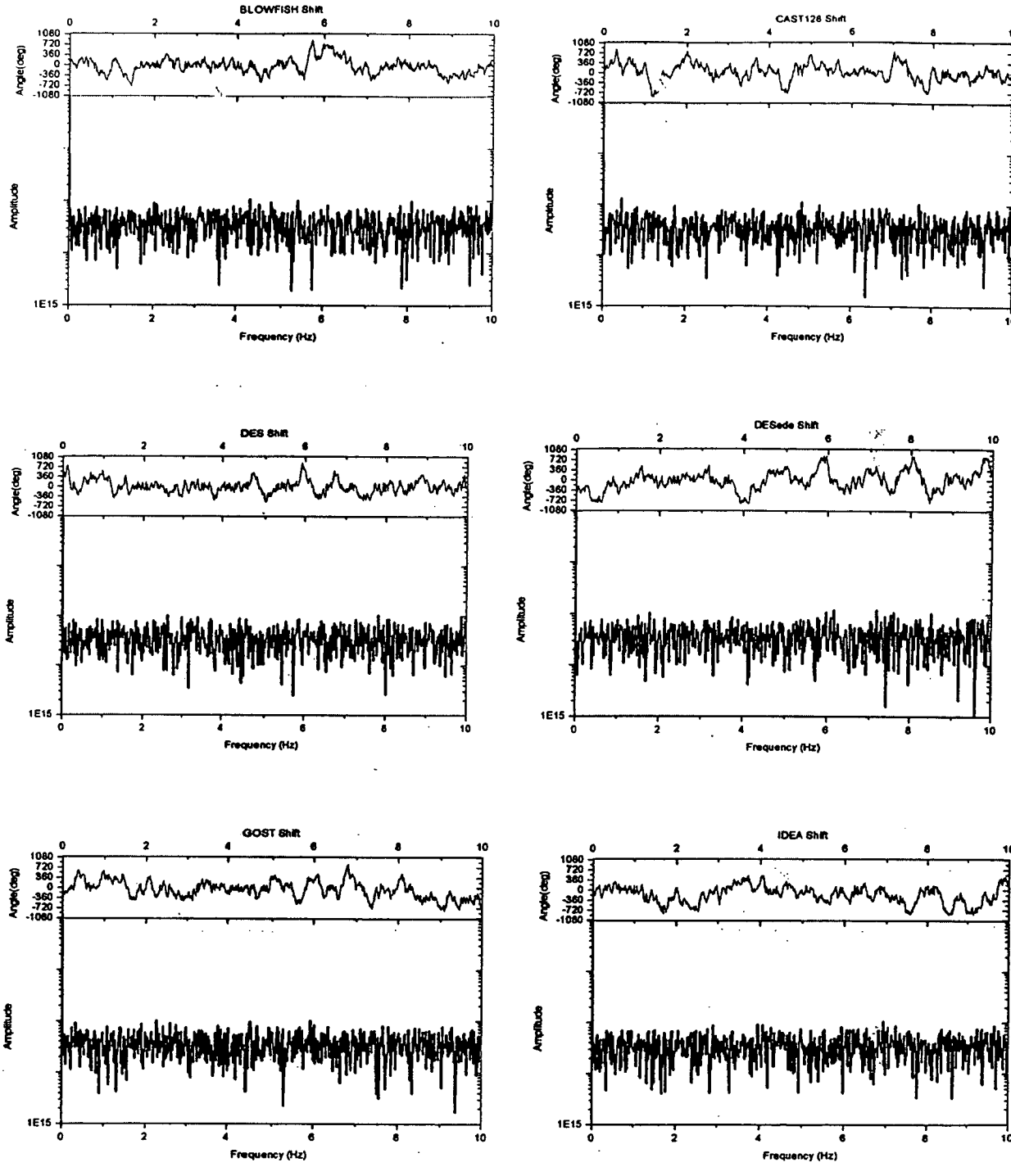
[그림 2-16]는 암호알고리즘에 의해서 평문을 암호문으로 변형한 결과에 대한 return map 이다.



[그림 2-16] Return map

나. 스펙트럼 분석

[그림 2-17]는 [그림 2-14]의 스펙트럼이다. 이 역시 시계열과 마찬가지로 특별한 주기성분을 찾을 수 없다. 즉, 암호 알고리즘의 예측 불가능함을 보여준다.

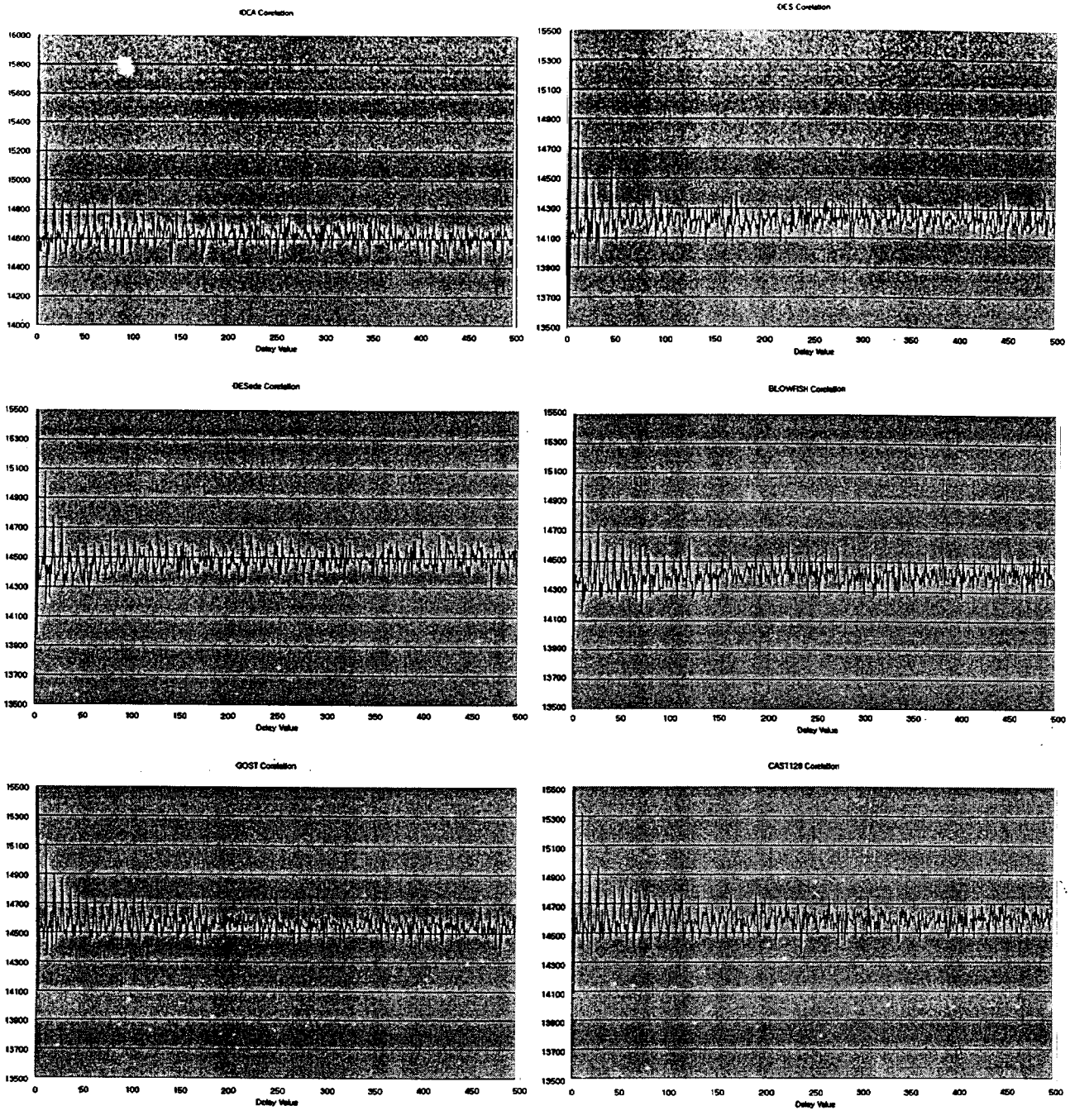


[그림 2-17] 스펙트럼 분석결과

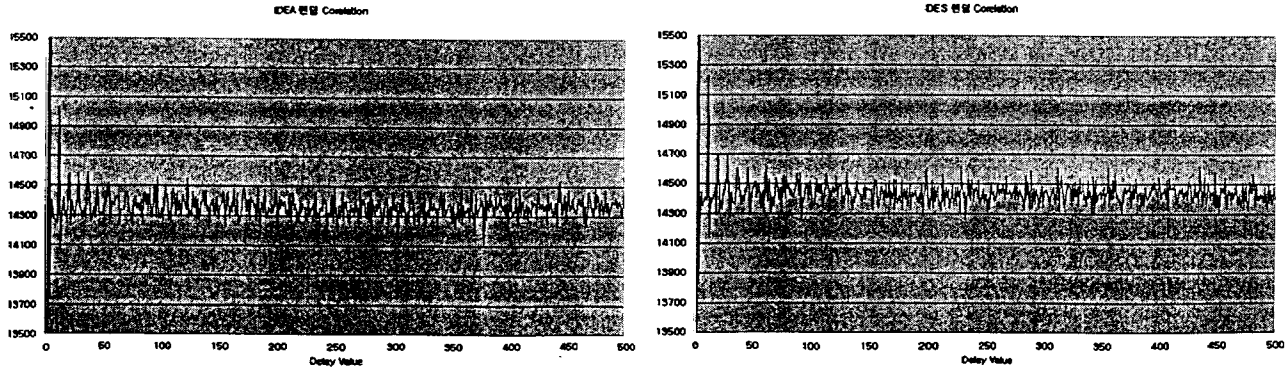
다. Correlation 분석

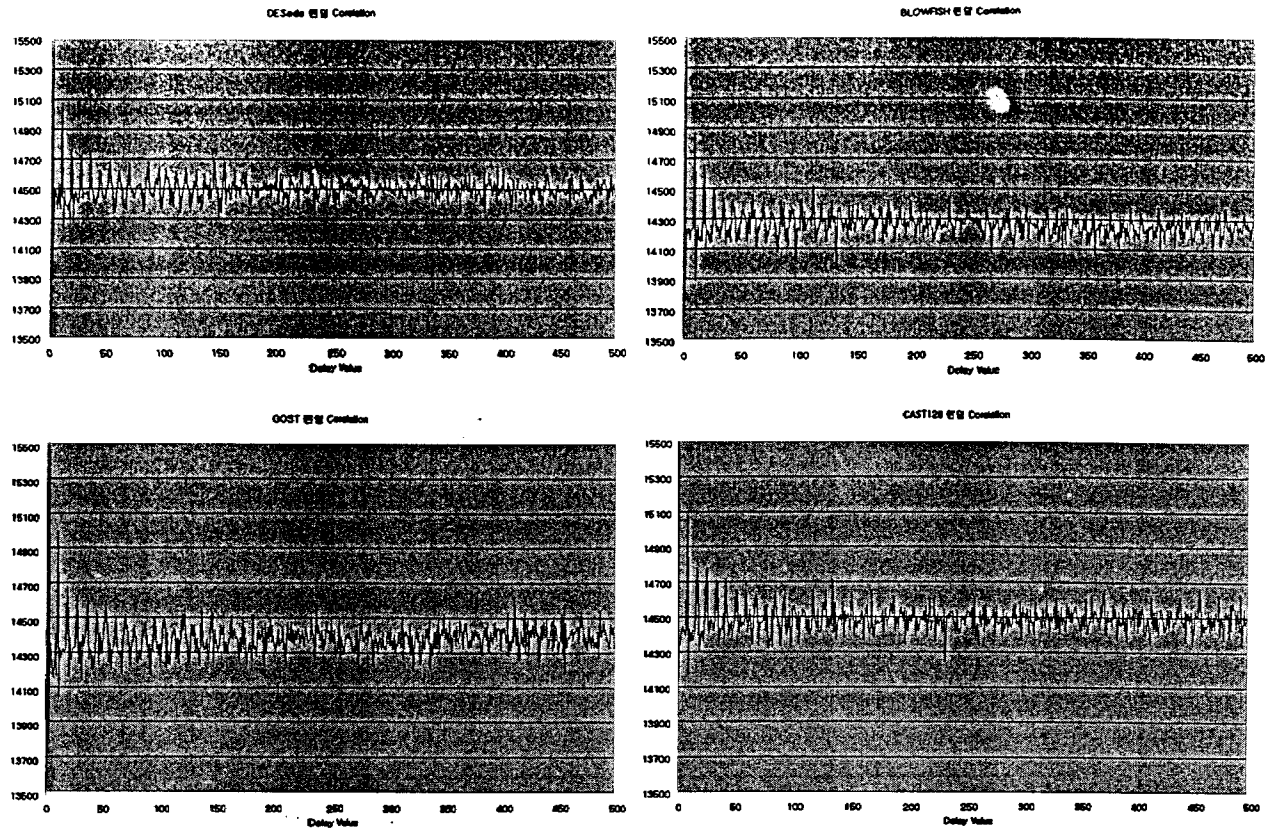
본 연구에서는 앞에서 살펴본 바와 같이 암호화 알고리즘 시스템은 시계열 분석에서나, 스페트럼 분석에서 특별한 주기성을 발견할 수 없었다. 그러므로 여기서는 Correlation평가를 통한 암호 알고리즘의 분석하고자 한다.

[그림 2-18]과 [그림 2-19]은 [그림 2-14]와 [그림 2-15]의 Correlation을 나타낸 그래프이다.



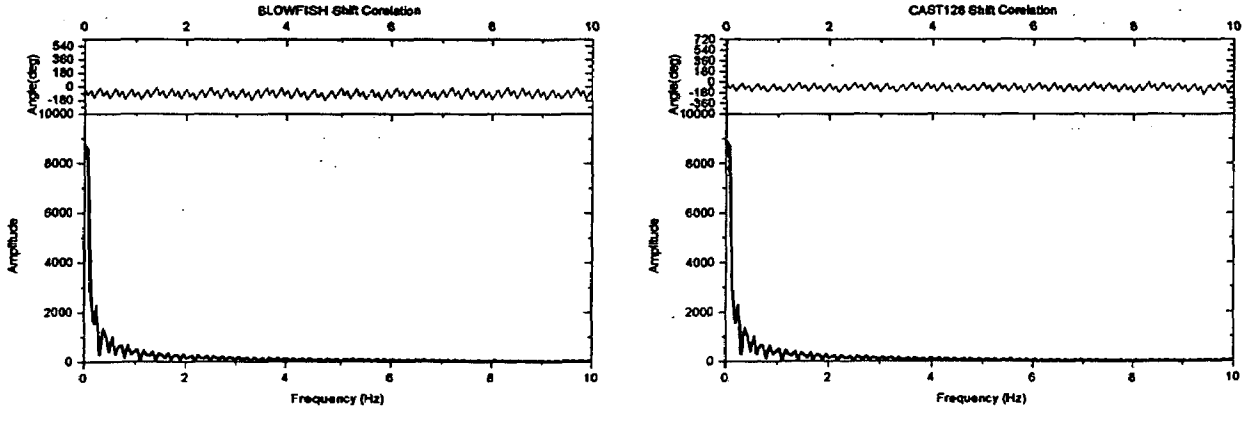
[그림 2-18] 순차증가 correlation 분석

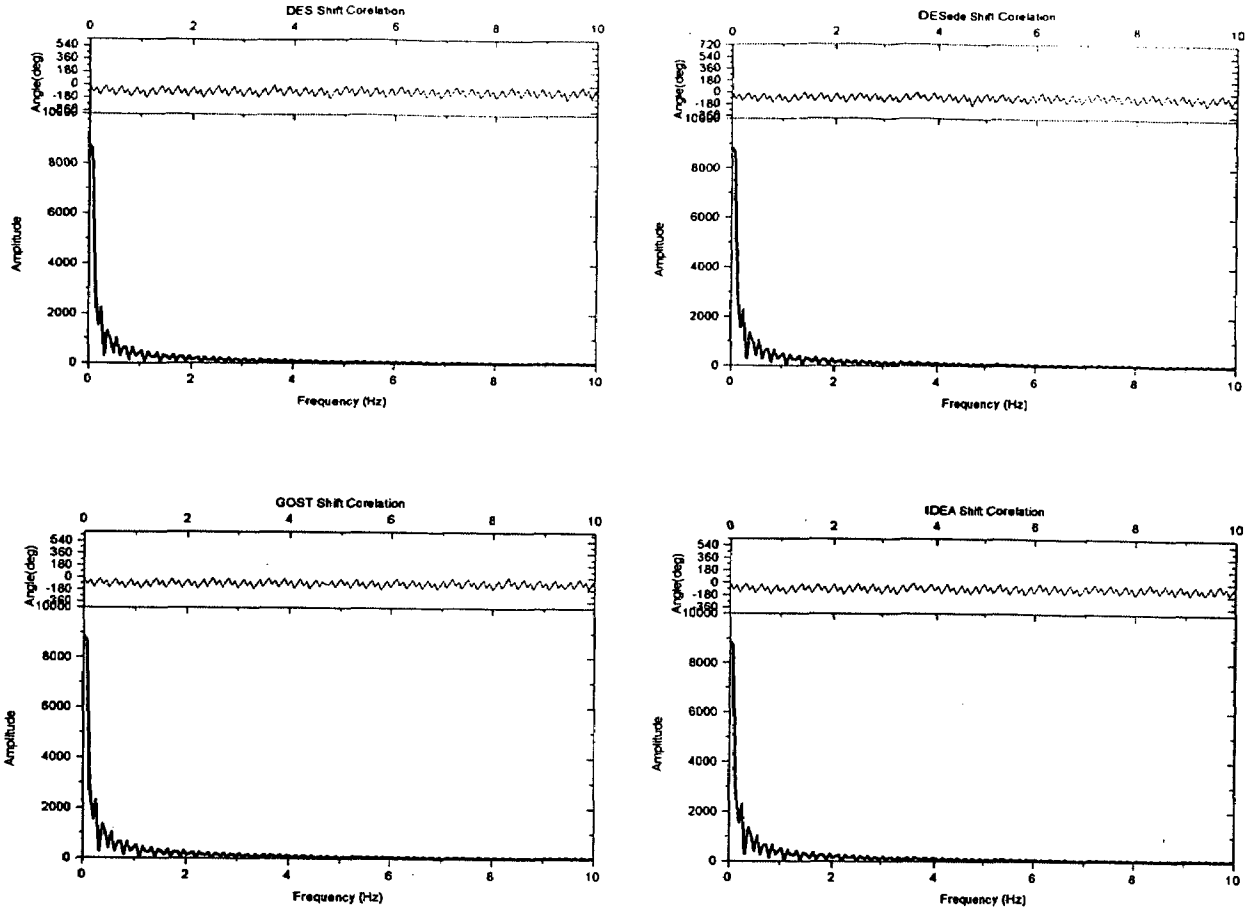




[그림 2-19] 랜덤 증가 correlecation

[그림 2-18]과 [그림 2-19]에서 살펴본 바와 같이 Correlation 그래프는 약간의 주기성을 보여주고 있다. 그러므로 이 주기성을 확인하기 위하여 [그림 2-18]의 스펙트럼 분석을 시도하여 [그림 2-20]에 나타내었다.





[그림 2-20] 스펙트럼 분석 결과

[그림 2-20]이 의미하는 것은 Correlation의 주 성분은 Autocorrelation이거나 Delay Value가 작은 경우에만 의미가 있음을 알 수 있다.

라. 암호문에서 평문의 파일 종류 추정

위에서 몇 가지 방법으로 암호화 알고리즘의 종류를 판별하기 위한 여러 가지 노력을 경주하였지만, 뚜렷한 방안이 나오지 않았다. 본 연구의 사전 연구시 프로그래밍 오류에 의한 데이터를 잘못 판단하여 특히 IDEA와 같은 경우, 주기성이 있는 것으로 추정하였다. 그러나 JAVA 암호화 라이브러리를 사용한 본 연구의 여러 결과를 통해 대칭형 암호화 알고리즘의 경우 전혀 주기성을 발견할 수 없었다.

그렇지만, 연구과정에서 나타난 데이터를 자세히 분석하면 평문 파일의 종류에 따라 암호문의 표준편차 분포가 따로 존재함을 알 수 있다. 이를 잘 이용하면 암호문의 일부분만을 취득하더라도 평문의 파일 종류를 추정할 수 있다.

본 연구에서는 암호화 알고리즘의 종류를 판별하는데는 성공하지 못했지만 어떤 종류의 파일이 암호화되었다 라는 것은 추정이 가능하였고, 이를 프로그램에서 구현하였다.

제 3장 평문 암호문 패턴 분석

1. 통계적 특성 분석에 의한 판별기술

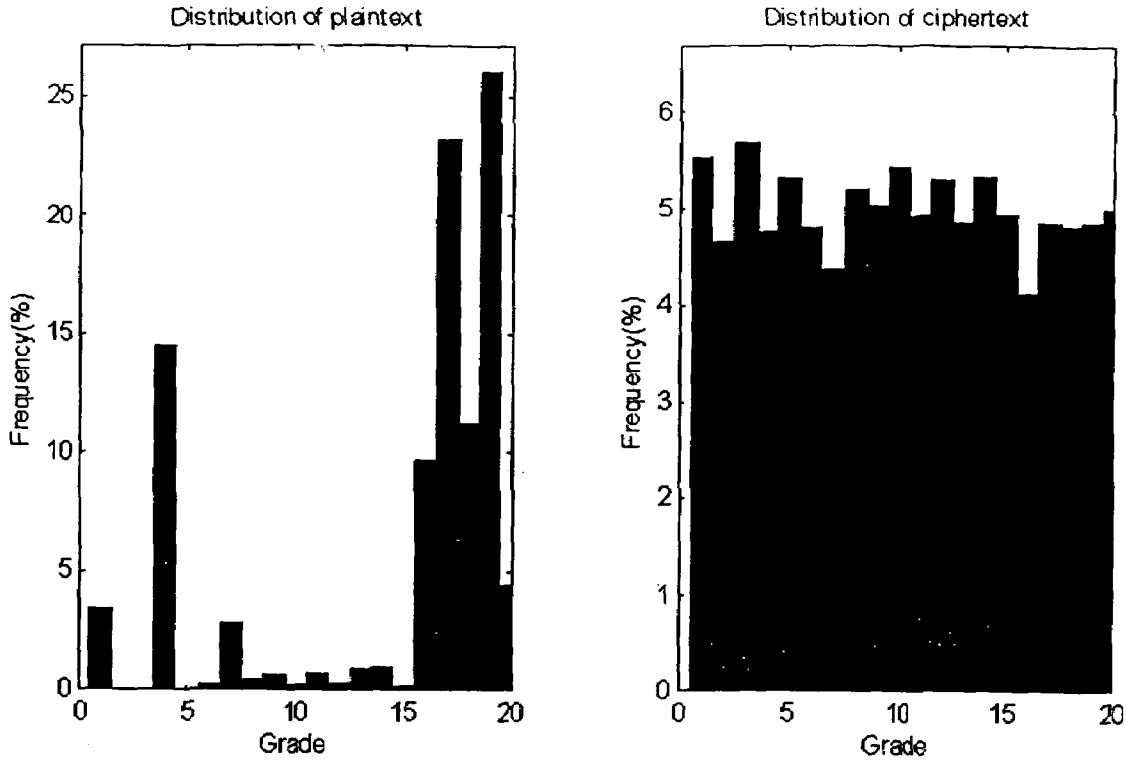
평문과 암호문은 각 문자에 대한 상대빈도의 차이에서 확인할 수 있다. 본 연구팀이 영문 3,000자로 이루어진 문장을 분석하였던 바, 각 문자는 다음과 같은 상대적인 빈도분포를 보였다. 이는 일반적인 영문자의 상대빈도분포 [Stallings, 1995]와 동일한 결과이다. 이에 비해 암호화된 문자의 상대 빈도분포는 <표 3-1>과 같다. 이는 평문과 암호문의 차이를 식별할 수 있는 가장 기본적인 방법이 된다.

알파벳	빈도수	비율	알파벳	빈도수	비율	알파벳	빈도수	비율
E	416	13.87%	D	141	4.70%	V	39	1.30%
R	294	9.80%	C	107	3.57%	B	37	1.23%
I	244	8.14%	L	95	3.17%	Y	35	1.17%
N	230	7.67%	G	73	2.43%	K	20	0.67%
A	214	7.14%	M	72	2.40%	X	16	0.53%
S	201	6.70%	F	68	2.27%	Q	3	0.10%
O	194	6.47%	U	65	2.17%	Z	1	0.03%
R	192	6.40%	P	53	1.77%	J	0	0.00%
H	144	4.80%	W	45	1.50%			

<표 3-1> 평문 3000글자 알파벳 빈도수 분포

문자 출현 빈도수에 의한 암호문/평문의 구별은 가장 손쉬운 방법이지만 여러 가지 트릭을 쓰는 경우 이를 피해갈 수 있게 된다. 따라서 일반적인 통계적인 기법 분석으로는 정확한 구분이 위와 같은 방법으로는 어렵다. [그림 2-1]에서와 같이 평문에서는 각 문자가 상대적인 빈도를 보이고 있으나 암호문에서는 각 문자가 고른 빈도의, 평균적인 분포를 보이고 있다.
실시간으로 암호문/평문의 식별을 위해서는 50자 이내의 짧은 송수신문에서 이를 구별해야만 한다. 본 연구진은 공동 연구기관인 한신대학교에서 수집한 암호화 알고리즘 20여개의 패턴을 조사한 바, 암호화된 암호문과 평문을 구별할 수 있는 통계적인 특징을 정의하였다.
첫째, 암호문의 경우 분포적인 특성이 고르게 나타나는 현상은 장문인 경우에 해당되나, 단문인 경우에도 특수문자의 발생 빈도가 영문자 알파벳과 크게 구분되지 않는다.
둘째, 평문의 문장을 3000자에서 단계적으로 1000자까지 감소시켰을 때에는 [그림 2-1]와 같은 빈도분포로 구분이 가능하나, 100자 이내의 짧은 문장에는 해당되지 않는다.

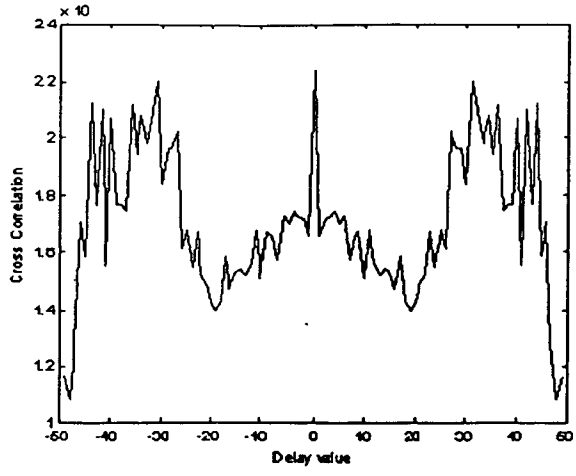
[그림 3-1]의 pdf분포는 메시지의 길이에 영향을 받기 때문에 메시지의 길이가 1000자 이상에서 활용성이 높은 것으로 나타났다. 메시지의 길이가 1000자 이내에서는 False alarm rate가 10%이상으로 높아지는 결과를 본 연구진은 확인하였다.

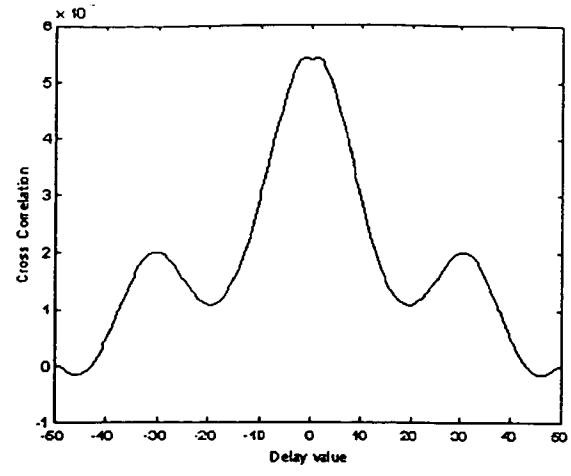


[그림 3-1] 평문(좌)과 암호문의 영문자 분포도

셋째, 암호문/평문의 통계적 특성을 분석하기 위하여 cross correlation을 이용한 임계치 설정을 실시한 결과 [그림 3-2]과 같이 암호문과 평문은 확연히 다른 특징을 갖고 있다. 이러한 특성은 메시지의 길이에 상관없이 거의 일정한 패턴을 갖고 있다는 점에서 이를 구분하기 위한 임계치로 활용이 가능하다.

[그림 3-2]는 암호문과 평문의 50개의 메시지를 이용한 cross correlation을 구한 결과로 correlation 함수의 차수를 보면 암호문의 경우에는 차수가 무한대로 변하지만, 평문의 경우에는 일정한 주기를 갖는 함수로 표현 가능하므로 이를 기준으로 암호문과 평문을 구별할 수 있다.





[암호문의 Cross Correlation] [평문의 Cross Correlation]

[그림 3-2] cross correlation을 이용한 임계치 설정

2. 암호화된 파일의 패턴 분석

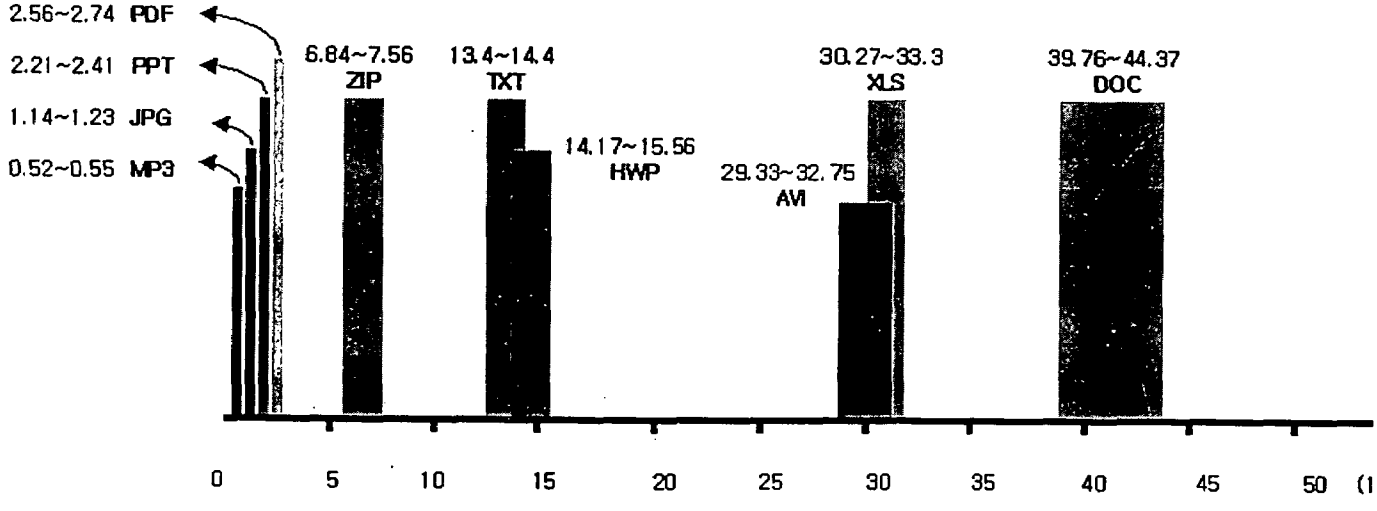
본 연구에서 주변에서 흔히 볼 수 있는 파일의 종류별로 ASCII값에 대한 돗수분포와 이들의 기대값을 조사하였다. 이를 통해 쉽게 평문과 암호문을 구분할 수 있었고 더 나아가 평문의 종류까지도 판별가능 하다고 생각한다. <표 3-2>를 보면 각각의 파일타입별 암호화문의 확률분포의 표준편차값을 비교하면 그 값이 파일타입별로 특색을 나타냄을 알 수 있다. 이에 이 값들의 평균과 표준편차를 구하면 <표 3-3>으로 표시할 수 있으며 평균값과 표준편차를 이용하여 기대영역을 구하여 볼 수 있다. [그림 3-3]은 이 영역을 그래프로 나타낸 것이다. 이러한 분석결과를 이용하면 우리는 임의의 암호문을 분석하여 비록 그 암호화된 알고리즘은 판단할 수 없지만 암호화된 소스의 파일타입이 무엇인가를 판별할 수 있을 것이다.

	blowfish	cast128	idea	desede	gost	des	평문
text	0.001434	0.001545	0.001359	0.001389	0.001308	0.001372	0.011207
pdf	0.000276	0.000263	0.000260	0.000264	0.000274	0.000250	0.004658
ppt	0.000218	0.000218	0.000232	0.000238	0.000232	0.000245	0.018366
xls	0.003061	0.003050	0.003132	0.003350	0.003049	0.003418	0.030858
doc	0.004039	0.004052	0.003996	0.004551	0.004091	0.004508	0.032008
hwp	0.001451	0.001448	0.001439	0.001477	0.001639	0.001462	0.026891
zip	0.000762	0.000692	0.000777	0.000701	0.000690	0.000691	0.011207
mp3	0.000049	0.000054	0.000052	0.000053	0.000054	0.000054	0.003663
jpg	0.000121	0.000123	0.000111	0.000119	0.000112	0.000118	0.002089
avi	0.002986	0.002984	0.002971	0.003348	0.002989	0.003342	0.008115

<표 3-2 확률분포의 표준편차>

	암호화 알고리즘 통합		기대값 범위	
	평균	표준편차	MIN	MAX
txt	0.0013	0.000052	0.0013	0.00143
pdf	0.0002	0.000009	0.0002	0.00027
ppt	0.0002	0.000010	0.0002	0.00024
xls	0.0031	0.000164	0.0030	0.00334
dos	0.0042	0.000252	0.0039	0.00445
hwp	0.0014	0.000076	0.0014	0.00156
zip	0.0007	0.000039	0.0006	0.00075
mp3	0.00005	0.000001	0.00003	0.00005
jpg	0.0001	0.000004	0.0001	0.00012
avi	0.0031	0.000187	0.0029	0.00329

<표 3-3 암호화 알고리즘의 통합, 기대값 범위>

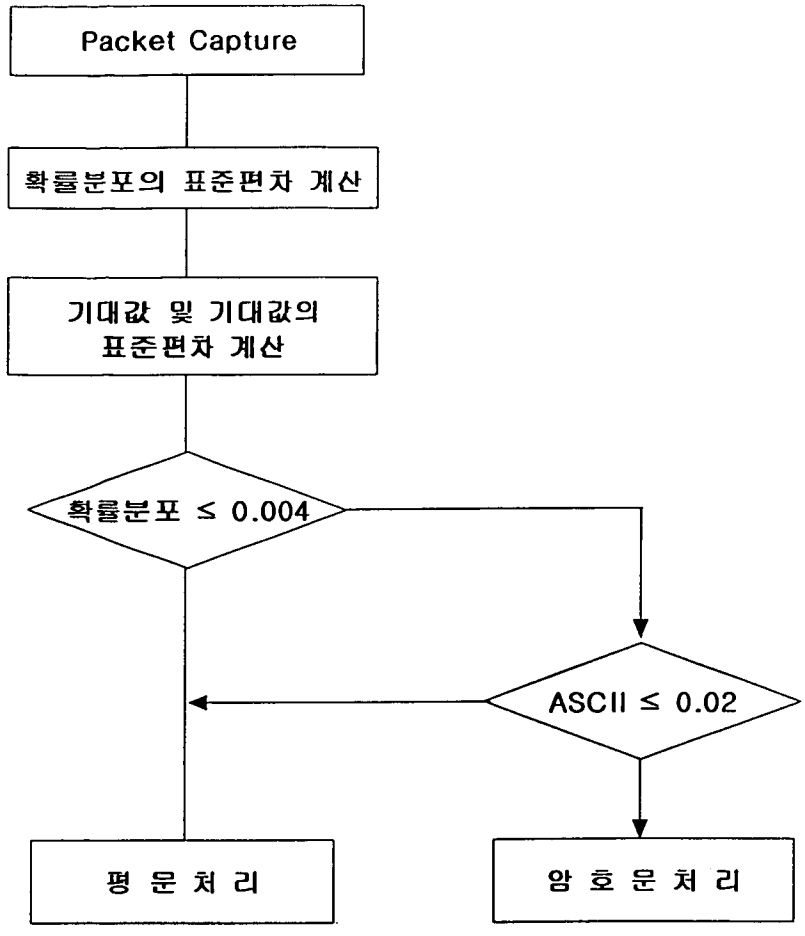


[그림 3 - 3] 암호화된 데이터의 표준편차의 기대값 영역

각종 암호 알고리즘에 의한 암호문의 ASCII 코드의 빈도수를 확률분포로 나타낸 것이 <표 3-2>이며 이를 통하여 평문과 암호문을 구분하는 알고리즘을 [그림 3-4]에 나타내었다.

가. 암호문 처리 루틴

다음의 처리 루틴은 LAN에서의 패킷을 수집하여 데이터를 조사, 그것이 암호문인지 평문인지를 구별하는 루틴이다.



[그림 3-4]평문 / 암호문 판별 알고리즘

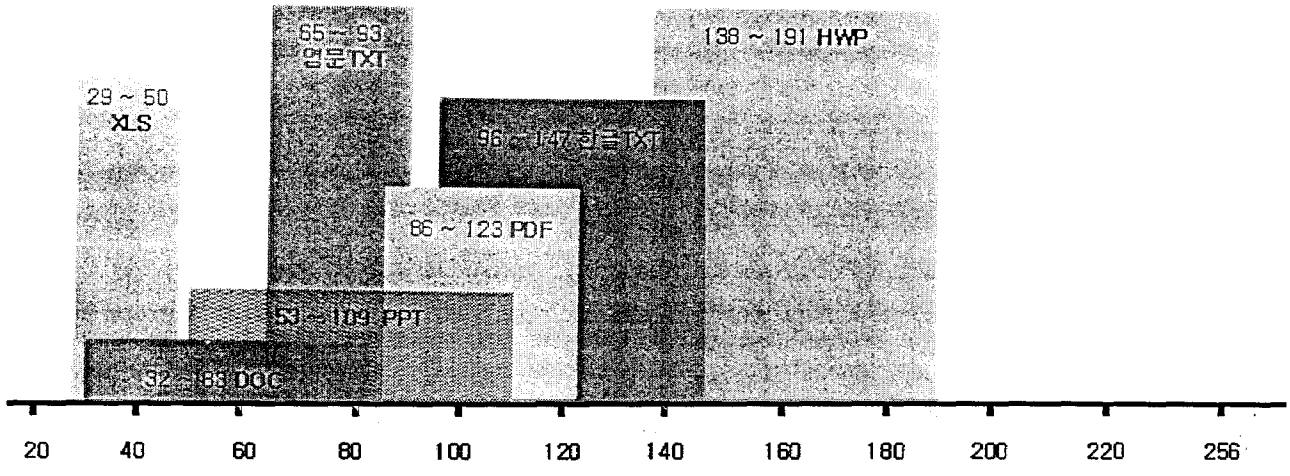
나. 평문처리 루틴

일단 평문이라고 판단이 될 경우, 보통 자주 사용되는 파일의 종류(아래한글, 워드, 엑셀 등등)를 구분하는 기능을 설명한다.

파일의 종류별로 ASCII 코드의 확률 분포에 대한 기대값의 평균, 표준편차(<표 3-3>, [그림 3-4]참조)를 이용하여 예상되는 파일의 종류를 아래의 단계에 따라 구분한다.

	평균	표준편차	아스키 코드값 빈도수				
			1순위	2순위	3순위	4순위	5순위
pdf	104.901	18.1933	32	48	13	47	49
txt한글	122.459	25.4714	32	45	192	10	13
ppt	81.5741	28.4256	0	1	255	32	2
hwp	164.777	26.6606	255	0	32	219	97
doc	57.6440	25.4898	0	255	32	102	1
xls	39.6900	10.9556	0	255	2	1	32
txt영문	79.8622	13.9578	32	101	116	97	110

<표 3-4> ASCII 코드에 확률 분포에 대한 기대값의 평균, 표준편차

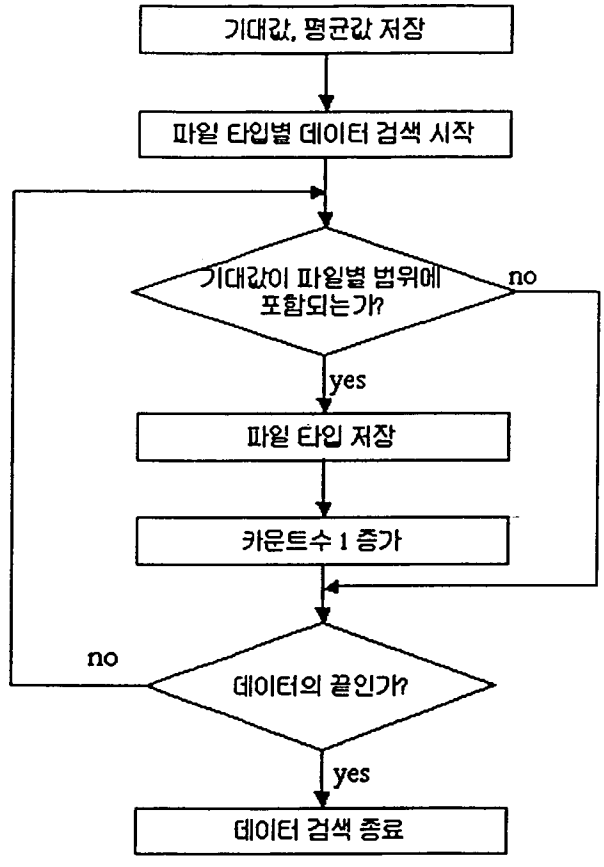


[그림 3-5] ASCII 코드 확률분포 기대값의 영역

제 1 단계

평문처리는 주어진 데이터 값을 이용하여 분석한다. 일차적으로 암호문인지 아닌지를 위의 루틴에서 판단한 후 암호문이 아닌 데이터를 평문으로 간주하고 분석에 들어간다.

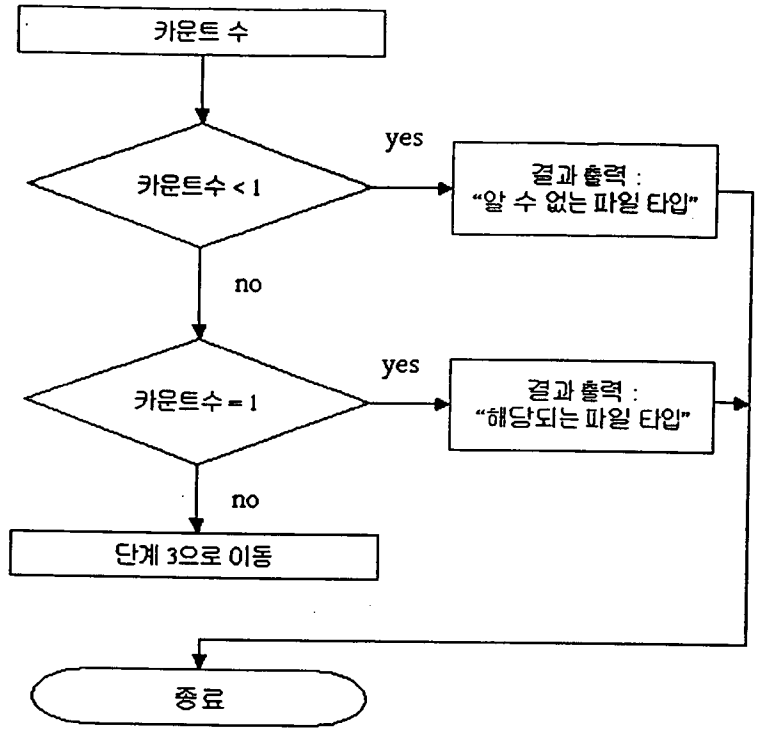
첫 번째 단계에서는 평문으로 간주된 데이터의 기대값과 평균값을 세팅한다. 그 후 기대값이 각 파일 타입의 범위에 속하는지 검색한다. 범위에 포함되면 그 때의 파일 타입을 저장하고 카운트를 증가한다.



[그림 3-6] 평문처리 1단계

제 2 단계

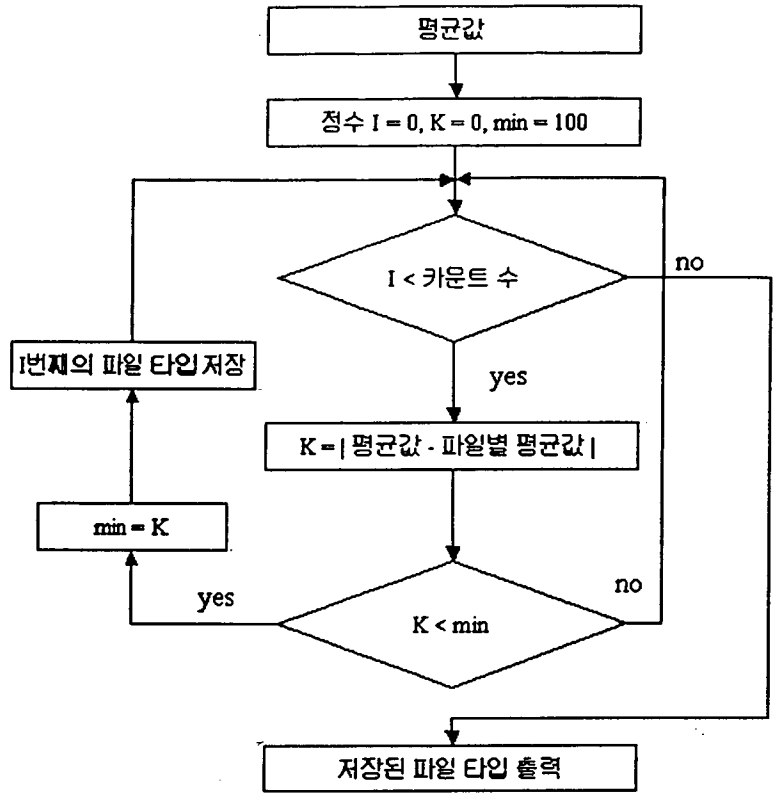
1단계에서 처리되어진 결과값을 이용한다. 즉 카운트수를 검사하여 카운트 수가 0이면 데이트가 알고 있는 파일 타입이 아니다. 그러므로 결과값은 "알 수 없는 파일 타입"으로 한다. 만약 카운트수가 1이면 여러 타입에 걸쳐 있지 않는 유일한 타입이므로 그 때에 해당하는 파일 타입을 출력한다.



[그림 3-7] 평문처리 2단계

제 3 단계.

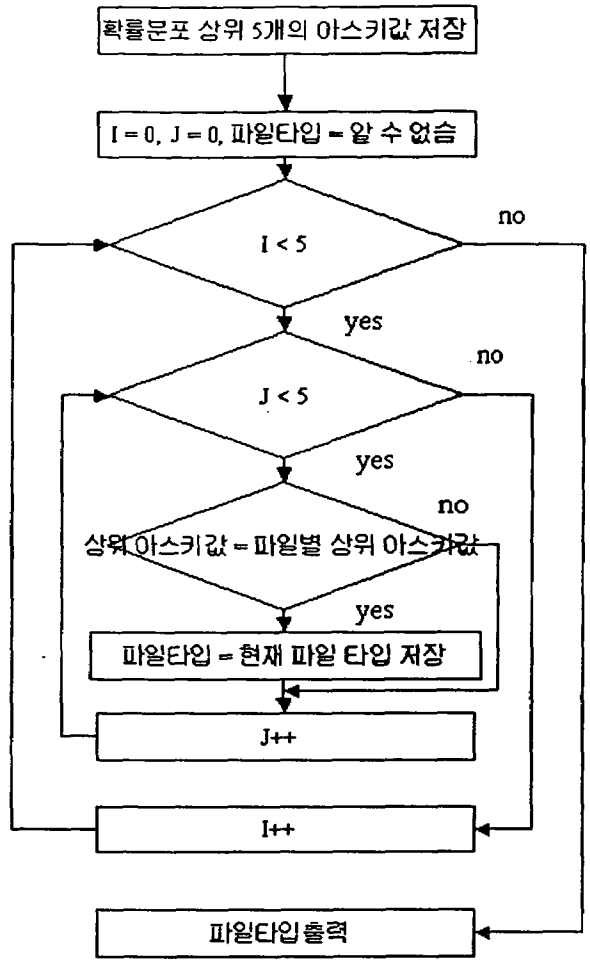
1단계에서 처리되어진 카운트수가 2이상일 경우에는 적정한 파일 타입을 선택하기 어렵다. 그래서 파일 타입을 결정하기 위한 선택작업을 해야한다. 우선 초기에 세팅한 평균값을 이용한다. 획득한 데이터의 평균값과 각 파일 타입별 평균값을 비교하여 그 차이의 절대값이 최소인 것을 구한다.



[그림 3-8] 평문처리 3단계

제 4 단계

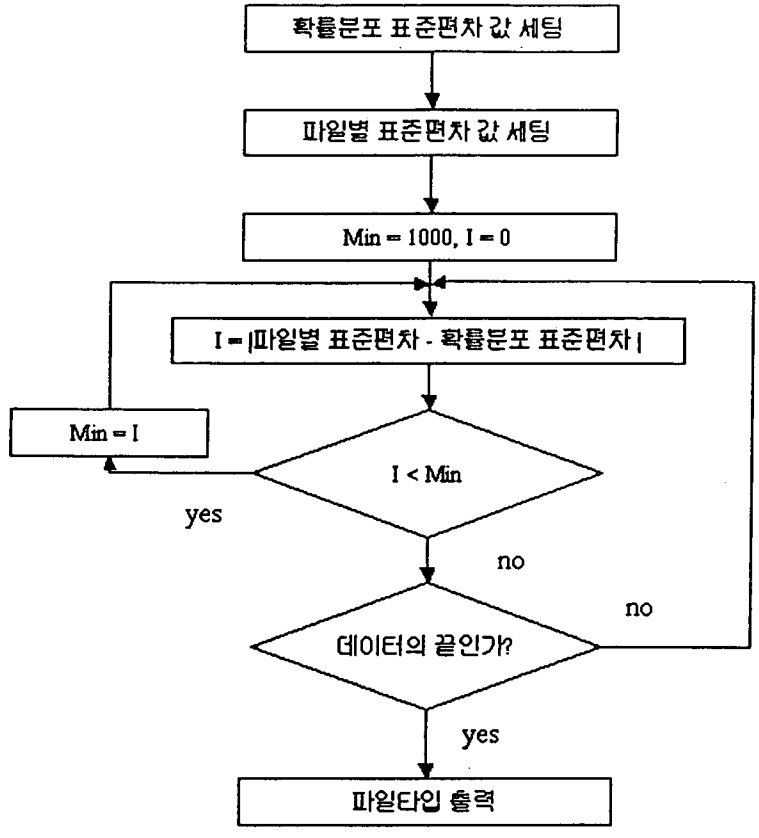
평균값 이외에 상위 5 순위의 아스키값을 1순위부터 비교하여 일치하는 파일 타입을 구한다



[그림 3-9] 평문처리 4단계

제 5 단계

이번 단계는 확률분포의 표준편차를 이용하여 각 파일 타입별 확률분포 표준편차와 비교하여 그 차이의 절대값이 가장 최소인 값의 파일 타입을 구한다.



[그림 3-10] 평문처리 5단계

제 6 단계

만일 3, 4, 5단계에서도 파일 타입이 2개 이상 중복되어 얻어진다면 3, 4, 5각 단계별 가장 많이 얻어지는 파일 타입을 1위로 하고 그 다음을 2위로 하여 결과를 출력한다.

출력 예) "아래아 파일일 경우가 가장 높으며 차선으로 한글텍스트 파일입니다."

제 4장 암호화 분석 시스템의 개발

1. 암호화 프로그램의 개발

가. 암호화 알고리즘의 구분

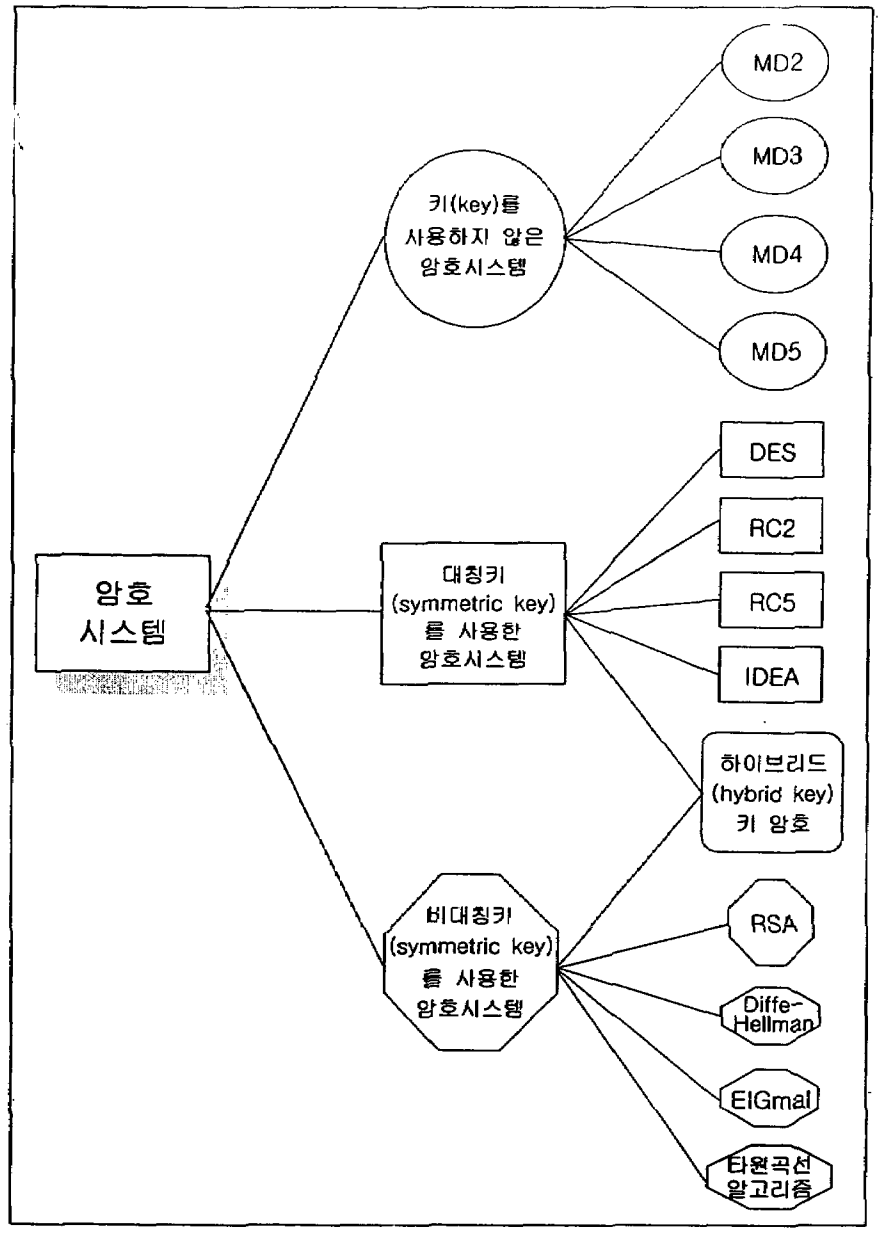
암호(Cryptography)를 평문을 해독 불가능한 형태로 변형하거나 또는 암호화된 통신문을 해독 가능한 형태로 변환하기 위한 원리, 수단, 방법 등을 취급하는 기술 또는 과학이다.

암호학에 관련된 중요한 용어 몇 가지를 설명하면 <표 4-1>과 같다.

용 어	번 역	설 명
plaintext 또는 cleartext	평문	전달해야 할 내용
ciphertext	암호문	암호화한 내용
encryption 또는 encipher	암호화	평문을 암호문으로 바꾸는 것
decryption 또는 decipher	복호화	암호문을 평문으로 바꾸는 것
symmetirc key	대칭키	평문을 암호문으로 암호문을 평문으로 생성하는 키 값이 같다
public key	공개키	평문을 암호문으로 생성하는데 사용되는 키 값이다. 비밀키와 항상 쌍을 이룬다
private key	비밀키	암호문을 평문으로 생성하는데 사용되는 키 값이다. 공개키와 항상 쌍을 이룬다
cryptography	cryptography	전달 내용의 보안을 연구하는 학문
cryptographer	cryptographer	cryptography를 수행하는 사람
cryptanalysis	cryptanalysis	암호문의 해독을 연구하는 학문
cryptanalyst	cryptanalyst	cryptanalysis를 수행하는 사람
cryptology	cryptology	cryptography와 cryptanalysis를 포함하는 수학의 한 분야
cryptologist	cryptologist	cryptology를 연구하는 사람
cryptographic algorithm 또는 cipher	cryptographic algorithm	암호화와 복호화를 위해 사용하는 수학 함수, 일반적으로 암호화를 위한 함수와 복호화를 위한 함수 두 개로 이루어진다.

[표 4-1] 암호화 알고리즘 관련 용어의 정의

[그림 4-1]에서 보듯이 암호 시스템은 평문을 암호문으로, 암호문을 평문으로 바꾸는 과정에 키(Key)의 사용유무와 종류에 따라서 크게 3가지 범주로 묶을 수 있다.



<표 4-2> 암호화 알고리즘의 구분

(1) 키(Key)를 사용하지 않은 암호 알고리즘

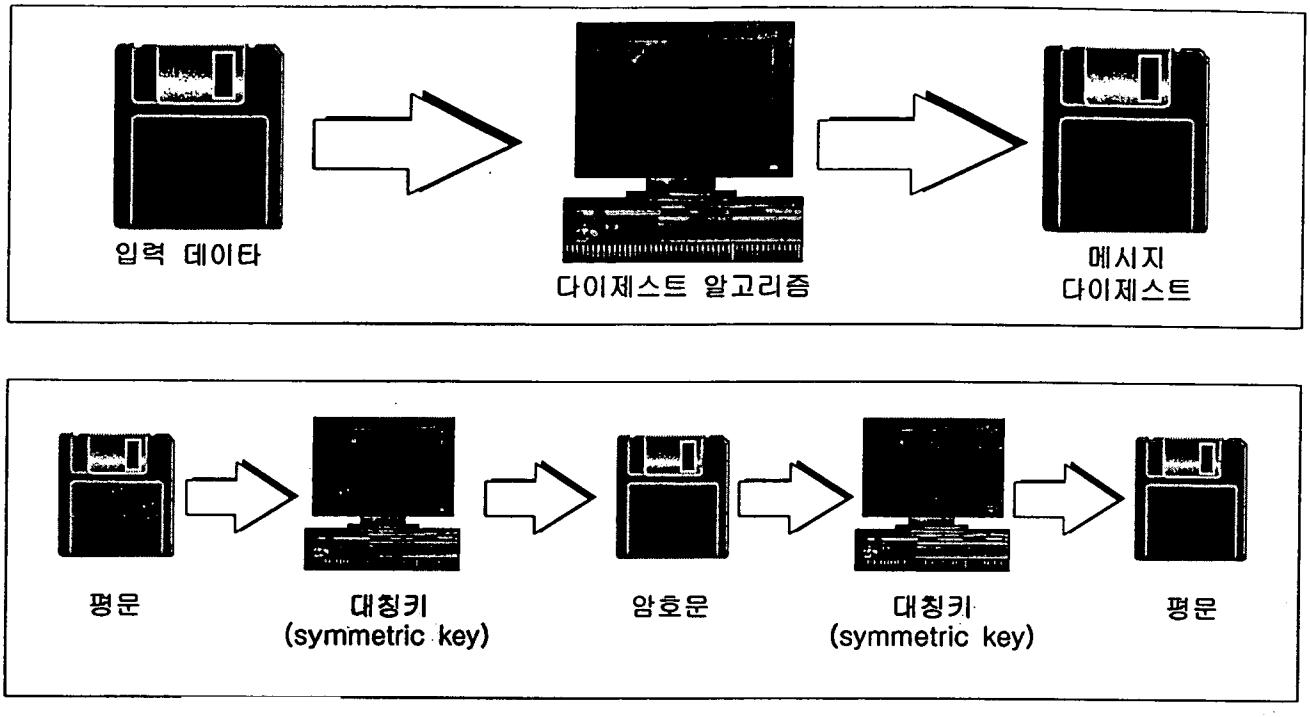
제목 그대로 특별한 Key 값을 사용하지 않고 평문을 암호문으로 바꾸는 알고리즘이다. 대표적으로 해쉬(hash function)와 메시지 다이제스트(message digest)를 예로 들 수 있다. 그림 2는 메시지 다이제스트를 보인 것이다.

메시지 다이제스트 알고리즘은 단방향 함수(one-way function)로서 입력 데이터를 일정한 크기의 암호문으로 출력한다. 이 메시지 다이제스트에 관한 자세한 설명은 다음에 소개될 메시지 다이제스트의 구현 부분에서 상세히 다루겠다.

(2) 대칭키(symmetric key)알고리즘

대칭키 암호화 알고리즘은 Encryption과 Decryption에 하나의 키를 사용하는 것으로 대칭키 암호화 알고리즘의 종류는 DES, RC2, RC4, RC5, IDEA, Blowfish등이 있다. [그림 5-2]는 암호화와 복호화 과정에서 동일한 대칭키를 사용하는 대칭키 알고리즘을 나타낸다.

대칭키 암호 알고리즘의 장점은 비대칭키 암호 알고리즘에 비해 암호화하는 것과 복호화하는 시간이 빠르다는 점이다. 그러나 대칭키 암호 알고리즘은 암호화하는 측과 복호화하는 측이 동일한 키를 사용하므로, 키를 아는 제3자가 암호문을 가로챈다면 보안 유지가 어려운 단점이 있다.



[그림 4-1] 암호화 알고리즘의 구분

(3) 비대칭키(asymmetric key)알고리즘

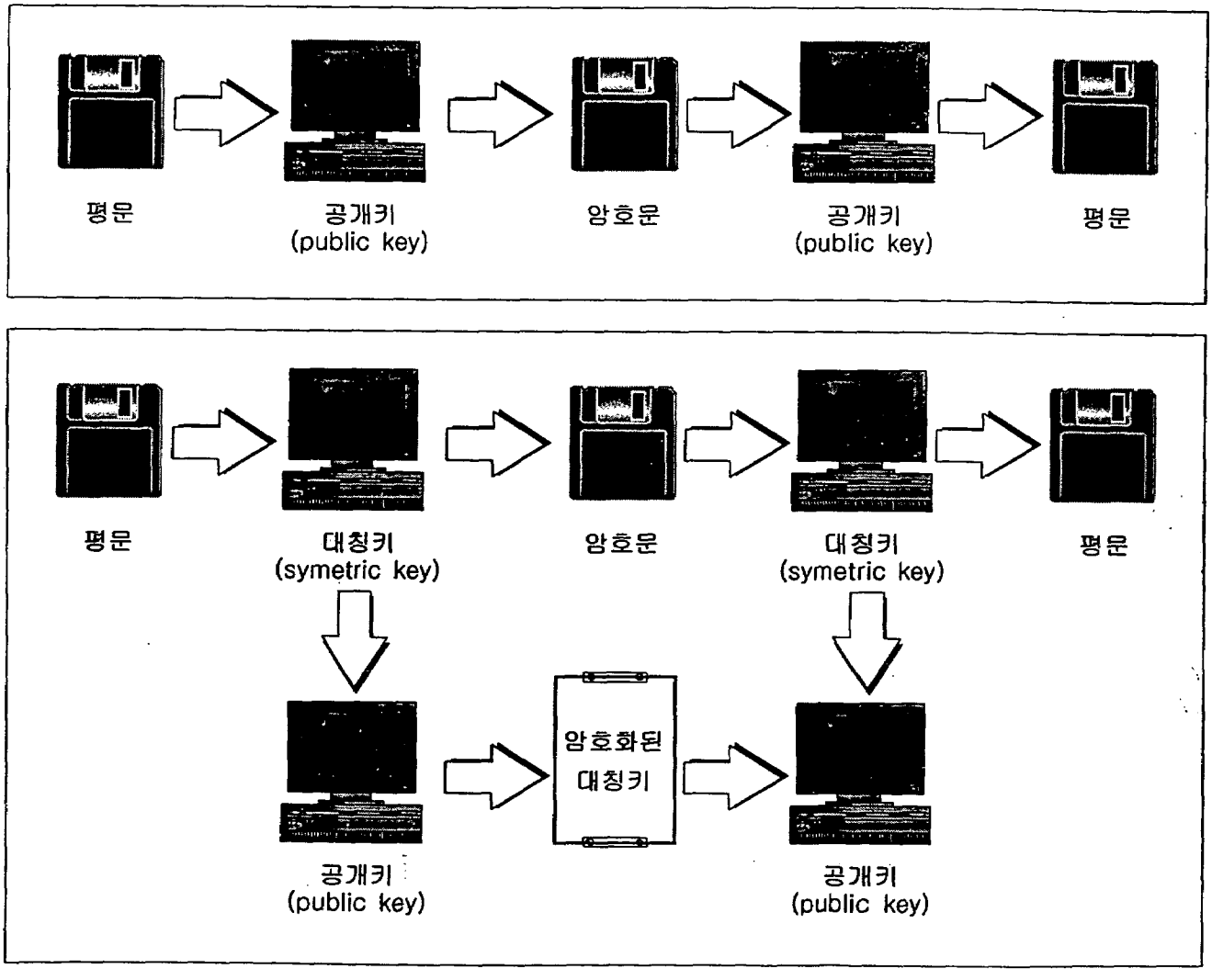
비대칭키 암호화 알고리즘은 서로 다른 두 개의 키를 이용하여 하나의 키는 암호화에, 다른 하나의 키는 복호화에 사용한다. 암호화와 복호화에 사용되는 키는 쌍을 이룬다. 비대칭키 암호화 알고리즘으로는 RSA(Rivest Shamir Adleman), Difflie-Hellman, ELGamal, 타원곡선 알고리즘 등이 있다. 그림 4는 비대칭키인 공개키와 비밀키를 이요한 암호 시스템을 나타낸다.

공개키와 비밀키는 수학적인 연산에 의해 생성되는데, 공개키는 뜻 그대로 모두에게 공개되어 있어서 암호문을 생성하고자 하는 사람은 모두 이 공개키를 사용할 수 있다. 그러나 공개키를 이용하여 생성된 암호문을 평문으로 복호화하기 위해서는 항상 비밀키가 필요하다. 이 비밀키는 개인만이 알고 있는 것이어서 암호가 유지될 수 있다. 이러한 비대칭키 암호 알고리즘은 암호화하는 키와 복호화하는 키가 서로 다르기 때문에 키 관리가 쉽고 암호의 안정성이 높게 평가된다. 반면 대칭키에 비해 연산 수행 시간이 오래 걸리는 단점이 있다.

앞서 살펴본 대칭키의 단점과 비대칭키의 단점을 보안한 하이브리드(hybrid) 암호 시스템이 현재 많이 응용되고 있다. [그림 4-2]는 평문을 대칭키로 암호화하고 대칭키는 비대칭키로 암호화하는 하이브리드 시스템을 나타낸다.

하이브리드 암호 시스템은 비대칭키의 단점인 시간적인 측면을 보완하기 위해 대칭키를 이용

하여 전달하고자 하는 평문의 암호문을 생성한다. 생성한 암호문을 상대방에게 전송할 때는 이 암호문을 복호화 하는데 필요한 대칭키를 비대칭키 암호 알고리즘을 사용하여 암호화한 후 전달한다. 수신측은 먼저 비대칭키 암호 알고리즘을 이용하여 대칭키를 복호화하고, 복호화한 대칭키를 이용하여 다시 암호문을 평문으로 복호화한다. 시간적인 측면을 고려해 보면 대칭키 알고리즘에 사용되는 키의 길이는 현재 128KB(최근 미국에서는 256KB 길이의 키를 사용하는 알고리즘이 소개되었다)를 넘지 않기 때문에 대칭키를 암호화하고 복호화하는 시간이 오래 걸리지 않는다. 그러므로 대칭키를 이용하여 전달하고자 하는 긴 문서를 암호화하고 복호화하는 것이 효율적이다.



[그림 4-2] 하이브리드 암호화 알고리즘의 작동

안전성 측면에서 보면 대칭키는 암호화하는 키와 복호화하는 키가 동일하므로 키를 전달하는 과정(암호문을 받은 수신측은 복호화 하기 위하여 대칭키가 필요하다)에서 다른 사람에게 누출된다면 안전을 보장할 수 없다. 하지만 하이브리드 암호 시스템은 비대칭키 암호 시스템을 이용하여 대칭키를 전달하므로 다른 시스템에 비하여 상대적으로 시간이 적게 걸리고 보다 안전하다고 할 수 있다.

2. 대칭키 암호 알고리즘

가. 대칭키와 대칭키 알고리즘

대칭키를 이용한 암호화 방법이란 하나의 키를 이용하여 주어진 데이터를 암호화하고, 동일한 키를 이용하여 암호문을 복호화해서 원래의 데이터를 생성하는것이다. 대칭키 암호 알고리즘에 사용되는 키를 대칭키 또는 비밀키라고 한다. 대칭키 암호 알고리즘은 대칭키를 생성하여 데이터를 암호화하므로 이전에서 소개한 메시지 암호 알고리즘보다 안정적이다. 본 연구에서 실행한 암호화 알고리즘은 다음과 같다.

RC2

RC2는 대칭키를 사용하는 블럭 암호 알고리즘으로 DES를 대체하기 위해 개발되었다. 입력 블록과 출력 블럭의 크기는 8바이트이며, 대칭키의 길이는 1~128바이트까지 다양하다. 보통 8바이트의 대칭키를 사용한다.

RC4

RC4는 다양한 길이의 키 크기를 가지는 스트림 암호 알고리즘이다. 알고리즘은 OFB 모드 형태로 작동하며 키의 스트림은 평문과는 독립적이다.

RC5

RC5는 워드 단위에 기초한 블럭 암호 알고리즘으로 블럭의 크기, 대칭키의 크기, 암호 알고리즘 적용 횟수와 같은 다양한 옵션을 가진다. 블럭 크기는 워드크기의 두 배이며, 일반적으로 16, 32, 64 등과 같이 사용된다. 암호 알고리즘의 적용 횟수는 최소 6회 이상이어야 안전성이 있다고 평가되고, 최소 12회 이상, 가능하다면 16회 정도를 권장한다.

IDEA

IDEA는 8바이트의 블럭과 16바이트의 대칭키를 지닌다. IDEA는 출력 변화에따라 8번의 암호 알고리즘을 적용하는 순환적인 암호 방법이다.

DES

대칭키 암호 알고리즘의 대명사격인 DES는 8바이트의 평문 또는 암호문과 8바이트의 대칭키를 지니는 블럭 암호 알고리즘으로 암호문 생성 알고리즘과 암호문 해독 알고리즘은 서로 역으로 작용한다. 8바이트의 비밀키 중에서 실제로사용되는 대칭키의 길이는 56비트이며 각 바이트의 최소 중요 비트(leastsignificant bits)는 에러 검사에 사용된다. 지난 15년 동안 내구력이 있었던DES는 이제는 수명이 끝나가고 있다. 56비트의 키 사이즈는 무차별공격(Brute-Force Attack)에 의해 키 값이 찾아질 수 있다.

DES3

DES3(또는 tripleDES, DES_EDE3)은 DES의 보안 효과를 증가시키기 위해 사용되며, 각 8바이트의 대칭키는 암호-복호-암호에 사용된다. 따라서 실제 키크기는 24바이트가 되고 암호문과 평문에 적용하는 데이터의 크기는 DES와 같이 8바이트이다.

DESX

DESX는 DES_EDE3 수준의 안정성을 지니면서 DES_EDE3의 단점인 속도를극복하고자 개발되었다. DESX는 DES_EDE3과 마찬가지로 실제 24바이트의크기를 가지는데, 평문은 첫 번째 8바이트 키와 XOR 연산을 하며 이 결과를 두 번째 8바이트 키로 암호화와 복호화를 한다. 이 결과를 세 번째 8바이트 키로 XOR 연산을 한다.

Blowfish

BLOWFISH는 DES를 대체하기 위한 새로운 대칭키 블럭 암호화 알고리즘으로 1993년 BRUCE SCHNEIER에 의해 만들어 졌다. 데이터의 블럭 사이즈는64비트를 이용하고, 대칭키의 크기는 32비트에서 448비트까지 적용이 가능하다.암호화 속도는 DES나 IDEA보다 빠르고 저작권이 무료라는 장점이 있다.

나. 대칭키 알고리즘의 구현

(1) 인터페이스를 다루는 클래스

다음은 java.security.Key 인터페이스로서 키 생성을 위한 모듈이다.

```
package java.security;

public interface Key extends java.io.Serializable {
static final long serialVersionUID = 6603384152749567654L; - ①
public String getAlgorithm();       - ②
public String getFormat();          - ③
public byte[] getEncoded();         - ④
}
```

① 사용자에게 고유 번호를 부여하여 자바 가상 머신이 이를 인식한다.
② 키 인터페이스를 선언하여 사용하는 암호 알고리즘의 이름을 돌려준다.
③ 키 값을 바이트의 배열로 매치시키는 것을 인코딩이라고 한다. 키를 인코드하는데 사용되는 방법의 이름을 돌려준다.
④ 키를 인코딩한 결과를 돌려준다.

java.security.Key 인터페이스는 모든 키의 특성을 선언한 것이다. 이것을 상속받는 다른 인터페이스들은 세분화된 키의 특성을 갖는다. java.security.Key 인터페이스를 상속받는 대표적인 인터페이스는 다음과 같다.

- java.security.PublicKey 인터페이스 : 공개키 알고리즘이나 전자서명에 사용되는 공개키와 비밀키의 쌍 중에서 공개키에 관한 정보를 담고 있는 인터페이스이다.
- java.security.PrivateKey 인터페이스 : java.security.PublicKey 인터페이스와 함께 공개키와 비밀키의 쌍 중에서 비밀키에 관한 정보를

담고 있다.

- java.security.KeyPair 클래스 : 쌍을 이루는 공개키와 비밀키에 관한 정보를 담고 있는 클래스이다.
- javax.crypto.SecretKey 클래스 : IAIK에서 제공하는 자바 암호 클래스를 설치했다면 설치한 암호 클래스 중에 javax.crypto.SecretKey 인터페이스가 구현되어 있다. 이 클래스는 대칭키 암호 알고리즘에 적용되는 대칭키에 관한 정보를 담고 있다.

javax.crypto.SecretKey 클래스를 이용한 자바 프로그램은 다음과 같다.

```
package javax.crypto;
import java.security.Key;
public abstract interface SecretKey extends Key {
}
```

SecretKey는 추상 클래스로 선언만 되어 있고 적용되는 대칭키 알고리즘마다 키의 특성이 달라 실제 내용은 서로 다르게 구현된다. 그러나 이름은 모두 SecretKey 객체로 생성되는데, 이것은 객체지향 언어의 특징을 잘 활용한 예이다.

(2) 키를 생성하는 클래스

자바 암호 알고리즘은 이를 위해 키를 생성하는 클래스를 각각 구현하는데 이런 클래스를 통틀어 키 생성(Key Generate) 클래스라고 한다. 알고리즘마다 키를 생성하는 방법은 다음과 같다.

① 키가 적용될 알고리즘을 위한 키 생성 객체를 만든다. 이 단계에서 키를 사용할 암호 알고리즘의 이름을 매개 변수로 전달한다.
② 키 생성 객체를 초기화한다. 초기화 방법은 적용하는 알고리즘마다 차이가있다. 대칭키 암호 알고리즘에서 사용되는 초기화 방법은 다음 장에서 설명하겠다.
③ 키 생성 객체의 generateKey 메소드를 호출하여 대칭키 암호 알고리즘을 위한 대칭키 또는 공개키 암호 알고리즘을 위한 키 쌍을 생성한다.

대칭키 암호 방식에 사용되는 키 생성 클래스는 javax. crypto.KeyGenerator 클래스이고, 공개키 암호 방식에는 java.secerity.KeyPairGenerator 클래스가 사용된다. 리스트 1은 KeyGenerator 클래스에 정의되어 있는 필드와 메소드들이다.

① KeyGenerator의 데이터를 선언하였다.
② KeyGenerator 클래스의 생성자이다. 생성자가 호출되면 앞서 선언한 필드에 데이터가 저장된다. 디폴트 생성자가 호출되는 경우 이 생성자가 별도의 getInstance 메소드와 init 메소드를 통해 필드에 데이터를 저장한다.
③ 객체가 가진 알고리즘 정보를 돌려주는 메소드이다.

④ getInstance 메소드를 이용할 알고리즘 또는 암호 클래스 제공자 정보를 객체에 저장한다.
⑤ 객체가 가진 암호 클래스 제공자 정보를 돌려주는 메소드이다.
⑥ init 메소드를 통해 키를 생성할 때 사용할 임의의 데이터나 키의 길이 등을 객체에 저장한다.
⑦ generateKey 메소드를 이용하여 생성한 대칭키를 돌려준다.

키를 생성하기 위해서는 KeyGenerator 객체를 초기화해야 한다. 객체를 초기화 할 때는 사용할 알고리즘에 따라 getInstance 메소드를 호출하고, 임의 값이나 키의 길이를 매개 변수로 하여 init 메소드를 호출하면 된다.

리스트 1 : javax.crypto.KeyGenerator

```
package javax.crypto

import java.security.InvalidAlgorithmParameterException;
import java.security.NoSuchAlgorithemException;
import java.security.NoSuchProviderException;
import java.security.Provider;
import java.security.SecureRandom;
import java.security.spec.AlgorithmParameterSpec;

public class KeyGenerator {
① private final KeyGeneratorSpi spi;
   private final Provider provider;
   private final String algorithm;

② protected KeyGenerator(KeyGeneratorSpi KeyGenSpi,
   Provider provider, String algorithm){}
③ public final String getAlforithm(){}
④ public static final KeyGenerator getInstance
   (String algorithm)
         throws NoSuchAlgorithmException {}
   public static final KeyGenerator getInstance
   (String algorithm, String provider)
         throws NoSuchAlgorithmException {}
         NoSuchProviderException {}
⑤ public final Porvider getProvider() {}
⑥ public final void init(SecureRandom random) {}
   public final void init(AlgorithmParameterSpec params)
      throws InvalidAlgorithmParameterException {}
   public final void init(AlgorithmParameterSpec params)
         SecureRandom random)
         throws InvalidAlgorithmParameterException {}
   public final void init(int strength) {}
   public final void init(int strength, SecureRandom random) {}
⑦ public final SecretKey generateKey() {}
}
```

java.secerity.KeyPairGenerator와 javax.crypto. KeyGenerator 클래스는 암호 알고리즘에 따라 서로 다른 객체를 생성하는 방법을 지원한다. 이것을 알고리즘에 따른 초기화라고 한다. 알고리즘에 따른 초기화 방법을 사용할 수 있는 것은 java.security.spec.AlgorithmParameterSpec 인터페이스 때문이다. 암호 알고리즘이 10개라면 맞는 키를 생성하기 위한 키 생성 클래스도 10개가 필요하다. 모든 암호 알고리즘은 AlgorithmParameterSpec 인터페이스를 상속받아 각 알고리즘을 구현한다.

다음은 대칭키를 이용한 DES 알고리즘 구현을 위해 키를 생성하는 방법이다.

```
KeyGenerator genkey = KeyGenerator.getInstance( "DES" ) ------ ①
genkey.init(new SecureRandom());  ------------------------------ ②
SecretKey seckey = genkey.generateKey()  ----------------------- ③
```

① KeyGenrator 객체 genkey의 getInstance 메소드를 호출할 때 매개 변수로 DES라는 스트링을 넘긴다. DES는 암호 알고리즘의 하나이다. 여기에 대칭키를 이용한 IDEA를 암호 알고리즘을 적용하려면 다음과 같이 바꾸면 된다.

```
KeyGenerator genkey = KeyGenerator.getInstance( "IDEA" );
```

암호 알고리즘이 바뀌어도 넘어가는 매개 변수만 바뀔 뿐 나머지는 그대로인데, 이것이 알고리즘에 따른 초기화이다.

② 대칭키를 생성하기 위해 필요한 임의의 값을 매개 변수로 전달하여 초기화한다.

③ generateKey 메소드를 이용하여 대칭키를 생성하고, seckey라고 선언된 변수에 대칭키 객체를 저장한다.

(3) 키의 전환

키의 전환은 생성한 키를 디스크에 저장하거나 네트워크 상으로 전달하는 방법에 관한 문제이다. 가장 보편적인 방법은 키를 바이트의 배열로 바꾸어 저장하는 것이다. javax.crypto.spec.SecretKeySpec 클래스나 javax.crypto.SecretKeyFactory 클래스, 또는 java.security.KeyFactory 클래스가 관여한다.

SecretKeySpec 클래스는 대칭키를 바이트의 배열로 바꾸는 가장 간단한 방법을 제공하는데, 다음 두 개중 하나의 생성자를 선언하여 사용한다.

```
public SecretKeySpec(byte[] key, String algorithm)
public SecretKeySpec(byte[] key, int offset, int len, String algorithm)
```

키를 바이트의 배열 또는 바이트의 배열을 키로 전환하는 것은 키의 생성절차와 유사하다. DES 알고리즘을 이용하여 대칭키를 바이트의 배열로, 바이트의 배열을 대칭키로 바꾸는 방법은 다음과 같다.

다음은 바이트의 배열을 원래의 DES 키로 바꾸는 메소드이다.

```
public SecretKey makeDESKey(byte[] input, int offset)
        throws NoSuchAlgorithmException, InvalidKeyException,
        InvalidKeySpecException{
SecretKeyFactory desFactory = SecretKeyFactory. - ①
        getInstance( "DES" );
KeySpec spec = new DESKeySpec(input, offset); - ②
return desFactory.generateSecret(spec); - ③
}
```

① SecretKeyFactory.getInstance()를 이용하여 DES 알고리즘의 키를 위한 Key 팩토리를 얻는다.

② 주어진 바이트 배열에서 DES 키를 나타낼 수 있는 KeySpec 객체를 생성한다.

③ SecretKeyFactory 클래스의 generateSecret 메소드를 이용하여 KeySpec 객체에서 SecretKey를 생성한다.

여기서 발생하는 예외 사항으로는 getInstance() 메소드를 사용할 때 일어나는NoSuch AlgorithmException이 있다. DESKeySpec 생성자에게 알맞은 길이의 바이트를 전달하지 못하면 InvalidKeyException 예외가 생긴다. generateSecret() 메소드를 사용할 때는 적절한 KeySpec 객체가 전달되지 않으면 InvalidKeySpecException이 발생한다.

다음은 DES를 바이트의 배열로 바꾸는 메소드이다.

```
public byte[] makeBytesFromDESKey(SecretKey Key)
        throws NoSuchAlgorithmException, InvalidKeySpecException {
          SecretKeyFactory desFactory = SecretKeyFactory.
          getInstance( "DES" );
        DESKeySpec spec = (DESKeySpec)desFactory.getKeySpec
          (key, DESKeySpec.class);
        return spec.getKey();
}
```

① SecretKeyFactory.getInstance()를 이용하여 DES 알고리즘의 키를 위한 Key 팩토리를 얻는다.

② 주어진 SecretKey로부터 DESKeySpec 객체를 생성하기 위해 getKeySpec메소드를 이용한다.
getKeySpec 메소드에 DESKey Spec 클래스를 전달하면 리턴되는 객체 타입은 DESKey Spec이다.

③ DESKeySpec 객체의 getKey 메소드를 이용하면 바이트의 배열을 얻을 수 있다.

(4) 패딩

블럭을 암호화할 때 주어진 데이타는 항상 정해진 블럭의 크기(대개 64비트)에 맞아떨어지는 것은 아니다. 즉 마지막 블럭의 데이터는 정해진 블럭 크기를 채우지 못할 수도 있다. 이 데이터를 그냥 암호화 하면 다시 복호화 할 때 실제 데이터의 마지막 부분을 알지 못하게 된다. 이 경우 블럭의 크기를 맞추기 위해 나머지 남는 부분을 유효하지 않은 값으로 채우는데 이것을 패딩(Padding)이라 한다. 암호문을 주고받을 때는 데이터의 패딩 정보도 일치해야 한다. 그래야 이것을 복호화하는 측에서 데이터를 복호화하고 패딩한 값을 제거할 수 있기 때문이다. 가장 많이 사용되는 패딩 알고리즘으로는 PKCS#5가 있다. 이것은 RSA 단체에서 공개키 암호 방식에 공식적으로 사용하는 방법이다.

PKCS#5는 정해진 크기의 블럭 중에서 비어있는 블럭의 개수로 비어있는 블럭을 채우는 방법이다. 즉 8바이트(=64비트)의 블럭 중에서 5개의 바이트 블록이 비어 있다면 숫자 5 값으로 5개의 빈 블럭을 채운다. 여기에 Cipher 클래스의 getPadding() 메소드를 이용한다. 이밖에 패딩을 위한 여러 알고리즘이 있다.

(5) 데이터 암호화

데이터를 암호화하기 위해 알고리즘을 정하고 알고리즘을 위한 키를 생성하고, 추가적인 운영 모드와 패딩 방법을 결정하고 나면 실제 데이터를 암호문으로 바꾸는 과정이 필요하다. 데이터를 암호화하거나 복호화 하는데는 Cipher 클래스가 사용된다. Cipher 클래스는 세 가지 방법(대칭키, 비대칭키, 하이브리드키)으로 다루어진다.

Cipher 클래스를 이용하여 암호문을 생성하는 것은 다음과 같은 세 단계를 거치게 된다.

```
Cipher cipher = Cipher.getInstance ( "DES/ECB/PKCS5Padding" ); - ①
   //getInstance 메소드를 이용하여 Cipher 객체를 생성한다.
cipher.init(Cipher.ENCRYPT_MODE, key); - ②
   //init 메소드를 이용하여 Cipher 객체를 초기화한다.
byte[] raw = cipher.doFinal(stringBytes); - ③
   //update 또는 dofinal 메소드를 이용하여 암호화 또는 복호화 한다.
```

Cipher.java 파일의 필드와 메소드 선언만을 모아 놓은 리스트 2를 보고 Cipher 객체를 생성하기 위한 각 단계를 자세히 살펴보자.

① Cipher 객체의 생성

Cipher 객체는 getInstance 메소드에 스트링 타입의 매개 변수를 전달하여 생성한다. 매개 변수로 전달되는 내용은 암호 알고리즘과 운영모드, 패딩 방법이다. 설치한 IAIK가 지원하는 암호 알고리즘은 RC2, RC4, DES, DESede, Blowfish가 있고 운영모드는 ECB, CBC, PCBC, GFBn, OFBn(여기서 n은 비트의 크기로 사용자가 지정한다. 지정하지 않으면 기존의 블럭 크기를 사용한다)이 있다. 그리고 패딩 방법으로는 NoPadding과 PKCS5Padding이 있다. 암호 알고리즘과 운영모드를 어떻게 사용하느냐에 따라 Cipher 객체의 초기화가 달라진다.

② Cipher 객체의 초기화

Cipher 클래스는 아래와 같이 정적 변수를 선언하여 사용한다.

```
public static final int ENCRYPT_MODE = 1, DECRYPT_MODE = 2;
```

이것은 init 메소드 호출시 이 객체가 데이터를 암호화하기 위한 것인지, 암호문을 복호화 하기 위한 것인지 명시하기 위해서이다. init 메소드를 이용하여 Cipher 객체를 초기화하는 경우 기본 적으로 이 모드에 관한 정보와 대칭키가 필요하다. 객체를 생성할 때 어떤 알고리즘과 운영모드 를 사용하느냐에 따라 초기화 방법이 다르다. DES/ECB/PKCS5Padding 방법을 사용하는 경우와 DES/CBC/ PKCS5Padding 방법을 사용하는 경우를 다음과 같이 비교할 수 있다.

```
Cipher cipher = Cipher.getInstance( "DES/ECB/PKCS5Padding" );
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] raw = cipher.doFinal(stringBytes);
```

DES/CBC/PKCS5Padding인 경우는 다음과 같다.

```
Cipher cipher = Cipher.getInstance( "DES/ECB/PKCS5Padding" );
IvParameterSpec spec = new IvParameterSpec(iv);
```

```
cipher.init(Cipher.ENCRYPT_MODE, key, spec);
byte[] raw = cipher.doFinal(stringBytes);
```

init 메소드를 이용하여 초기화하는 과정에는 IvParameterSpec 객체가 전달된다. CBC 운영 모드가 ECB와 달리 IV값을 필요로 하기 때문에 IvParameterSpec 객체를 생성하고 이것을 객체 의 초기화 과정에 이용한다.

③ update 메소드와 dofinal 메소드는 암호문을 평문으로, 평문은 암호문으로 바꾸는 역할을 한다. update 메소드는 호출될 때마다 전달되는 데이터를 누적하여 주어진 크기의 블럭이 차면 그 블럭을 암호문 또는 평문으로 전환하고, 블럭이 비어 있으면 블록을 채워줄 데이터를 기다린다. dofinal 메소드는 더 이상 데이터의 추가가 없음을 뜻하고 현재까지 저장된 데이터를 바로 암호화 또는 복호화한다.

④ 소개되어 있는 메소드는 Cipher 객체의 정보를 제공하는 메소드들이다. 각 메소드는 메소드의 이름처럼 사용자에게 객체가 가진 클래스 제공자, 알고리즘, 블럭 크기 등을 제공한다.

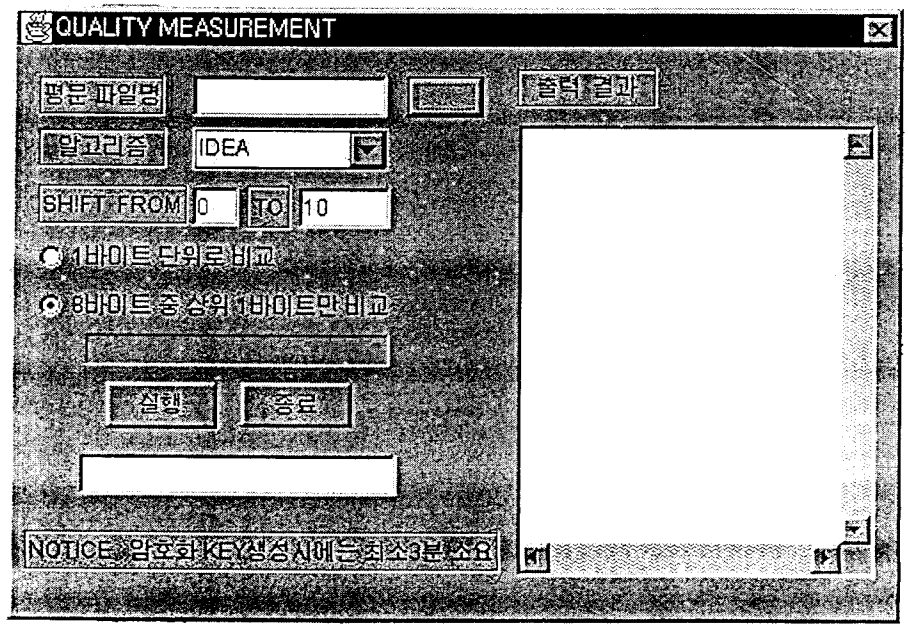
리스트 2 : javax.crypto.Cipher.java

```
package javax.crypto;
 public class Cipher {
    public static final int ENCRYPT_MODE = 1, DECRYPT_MODE = 2;
    private final CipherSpi spi;
    private final Provider  provider;
    private final String    transformation;
    protected Cipher(CipherSpi cipherSpi, Provider provider,
    String transformation) {}
    private static Object[] getCipherImplementation
        (String transformation, Provider p)
        throws NoSuchPaddingException {}
 ① public static Cipher getInstance(String transformation)
        throws NoSuchAlgorithmException, NoSuch
        PaddingException {}
    public static Cipher getInstance(String transformation, String
        provider)
        throws NoSuchAlgorithmException, NoSuchProvider
        Exception, NoSuchPaddingException {}
 ② public final void init(int opmode, Key key) throws
        InvalidKeyException {}
    public final void init(int opmode, Key key, SecureRandom
        random) throws InvalidKeyException {}
    public final void init(int opmode, Key key,
        AlgorithmParameterSpec params)
        throws InvalidKeyException, InvalidAlgorithm
            ParameterException {}
    public final void init(int opmode, Key key,
        AlgorithmParameterSpec params, SecureRandom random)
        throws InvalidKeyException, InvalidAlgorithm
        ParameterException {}
    public final void init(int opmode, Key key,
        AlgorithmParameters params)
        throws InvalidKeyException,
        InvalidAlgorithmParameterException {}
    public final void init(int opmode, Key key, AlgorithmParameters
        params, SecureRandom random)
        throws InvalidKeyException,
        InvalidAlgorithmParameterException {}
 ③ public final byte[] update(byte[] input) 소개ws
        IllegalStateException {}
    public final byte[] update(byte[] input, int inputOffset,
        int inputLen) throws IllegalStateException {}
    public final int update(byte[] input, int inputOffset,
        int inputLen, byte[] output)
        throws IllegalStateException, ShortBufferException {}
    public final int update(byte[] input, int inputOffset,
        int inputLen, byte[] output, int outputOffset)
        throws IllegalStateException, ShortBufferException {}
        public final byte[] doFinal()
        throws IllegalStateException,
        IllegalBlockSizeException, BadPaddingException {}
    public final int doFinal(byte[] output, int outputOffset)
     throws IllegalStateException, IllegalBlockSizeException,
        ShortBufferException, BadPaddingException{}
    public final byte[] doFinal(byte[] input)
        throws IllegalStateException, IllegalBlockSize
        Exception, BadPaddingException {}
    public final byte[] doFinal(byte[] input, int inputOffset,
        int inputLen)
        throws IllegalStateException, IllegalBlock
        SizeException, BadPaddingException {}
    public final int doFinal(byte[] input, int inputOffset,
        int inputLen, byte[] output)
     throws IllegalStateException, ShortBufferException,
        IllegalBlockSizeException, BadPaddingException {}
    public final int doFinal(byte[] input, int inputOffset,
        int inputLen, byte[] output, int outputOffset)
     throws IllegalStateException, ShortBufferException,
        IllegalBlockSizeException, BadPaddingException {}
 ④ public final Provider getProvider() {}
    public final String getAlgorithm() {}
    public final int getBlockSize() {}
    public final int getOutputSize(int inputLen) throws
        IllegalStateException {}
    public final byte[] getIV() {}
    public final AlgorithmParameters getParameters() {}
 }
```

① 한번에 읽어 들일 버퍼의 크기를 8192비트, 즉 1024바이트로 설정하였다.
② 프로그램 실행시 변수가 제대로 전달되는지 검사한다.
③ 현재 디렉토리에 대칭키를 가지고 있는 파일이 있는지 검사한다. 만약 이전에 대칭키를 만든 적이 없다면 예외사항이 발생하므로 다음에 있는 catch 구문으로 이동한다.
④ 현재 사용할 수 있는 대칭키가 없으므로 키를 생성한다.
⑤ 키를 생성하는 구문이다. 이미 설명한 내용이므로 생략한다. 이것이 이해가 된다면 DES 이외의 다른 알고리즘을 적용해 보자.
⑥ 암호문을 생성할 Cipher 객체를 생성하는 클래스이다. 역시 앞에서 설명했으므로 이해가 되면 운영모드와 패딩 방법을 바꿔 보자.
⑦ 파일을 실행할 때 전달받은 변수의 인덱스를 조사하여 암호화 할 것인지 복호화 할 것인지 결정하여 Cipher 객체를 초기화한다.
⑧ 읽어들일 파일과 기록할 파일을 열 스트림 객체를 생성한다.
⑨ Cipher 객체를 통해 생성된 암호문 또는 평문을 저장할 스트림 객체를 생성한다.
⑩ 읽어들일 데이터가 없을 때까지 데이터를 읽어들이면서 암호화 또는 복호화 작업을 하고 이를 다시 파일에 쓴다.

3. 암호화 알고리즘 분석 시스템

가. 평문과 암호문의 "QUALITY MEASUREMENT"분석 프로그램



[그림 4-3] 메인 화면

(1) APPLECATION 메인 클래스 정의

```
package cipherhigh;

import java.io.*;
import java.security.*;
import javax.swing.UIManager;
import java.awt.*;

public class CipherHighByteSigma {
  boolean packFrame = false;

  /**Construct the application*/
  public CipherHighByteSigma() {
    CipherHighByteSigmaFrame1 frame = new CipherHighByteSigmaFrame1();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    } else {
      frame.validate();
    }

    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height
frameSize.height) / 2);
    frame.setVisible(true);
  }
  /**Main method*/
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
```

```
    }
    new CipherHighByteSigma();
  }
}
```

(2) FORM을 만들고 데이터를 처리하는 클래스 정의

```
package cipherhigh;

import java.io.*;
import java.security.*;
import javax.crypto.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.beans.*;
import java.util.Vector;

public class CipherHighByteSigmaFrame1 extends JFrame {
  int available = 0;
  String fileName;
  JPanel contentPane;
  XYLayout xYLayout1 = new XYLayout();
  JButton jButton1 = new JButton();

  /**Construct the frame*/
  public CipherHighByteSigmaFrame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  /**Component initialization*/ // FORM 구성 부분
  private void jbInit() throws Exception  {
    jButton1.setForeground(Color.red);
    jButton1.setNextFocusableComponent(jTxtFld3);
```

```
        jButton1.setText("실행");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            jButton1_actionPerformed(e);
          }
        });
//setIconImage(Toolkit.getDefaultToolkit().createImage(CipherHighByteSigmaFrame1.class.getResource("[Your Icon]")))
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(xYLayout1);
        this.setResizable(false);
        this.setSize(new Dimension(477, 332));
        this.setTitle("QUALITY MEASUREMENT");
        jLabel1.setBorder(BorderFactory.createEtchedBorder());
        jLabel1.setText("평문 파일명");
        jLabel2.setBorder(BorderFactory.createEtchedBorder());
        jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
        jLabel2.setHorizontalTextPosition(SwingConstants.CENTER);
        jLabel2.setText("알고리즘");
        jLabel3.setBorder(BorderFactory.createEtchedBorder());
        jLabel3.setText("SHIFT  FROM");
        jLabel4.setBorder(BorderFactory.createEtchedBorder());
        jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
        jLabel4.setText("TO");
        jTxtFld2.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {

          }
        });
        jLabel5.setBorder(BorderFactory.createEtchedBorder());
        jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
        jLabel5.setText("출력  결과");
        jButton2.setForeground(Color.red);
        jButton2.setNextFocusableComponent(jComboBox1);
        jButton2.setActionCommand("");
        jButton2.setHorizontalTextPosition(SwingConstants.CENTER);
        jButton2.setText("찾기");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            jButton2_actionPerformed(e);
          }
        });
```

```
jTxtFld1.setNextFocusableComponent(jTxtFld2);
jTxtFld2.setNextFocusableComponent(jButton1);
jTxtFld1.setText("0");
jTxtFld2.setText("10");
jComboBox1.setNextFocusableComponent(jTxtFld1);
jTxtFld3.setNextFocusableComponent(jComboBox1);

jLabel6.setBorder(BorderFactory.createEtchedBorder());
jLabel6.setDisplayedMnemonic('0');
jLabel6.setText("NOTICE : 암호화 KEY생성시에는 최소3분 소요");
contentPane.setNextFocusableComponent(jTxtFld3);
jButton3.setForeground(Color.red);
jButton3.setText("종료");
jButton3.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jButton3_actionPerformed(e);
  }
});
jRadioButton1.setBorder(BorderFactory.createLoweredBevelBorder());
jRadioButton1.setText("1바이트 단위로 비교");
jRadioButton1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jRadioButton1_actionPerformed(e);
  }
});
jRadioButton2.setSelected(true);
jRadioButton2.setText("8바이트 중 상위 1바이트만 비교");
jRadioButton2.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jRadioButton2_actionPerformed(e);
  }
});
jFileChooser1.setApproveButtonText("");
jFileChooser1.setApproveButtonToolTipText("");
contentPane.add(jLabel3, new XYConstraints(12, 74, 80, -1));
contentPane.add(jLabel2, new XYConstraints(12, 44, 68, 21));
contentPane.add(jTxtFld3, new XYConstraints(94, 15, 104, -1));
contentPane.add(jLabel1, new XYConstraints(12, 16, -1, 21));
contentPane.add(jComboBox1, new XYConstraints(94, 43, 105, 23));
contentPane.add(jTxtFld1, new XYConstraints(94, 75, 25, 21));
```

```
    contentPane.add(jButton2, new XYConstraints(209, 15, 41, 22));
    contentPane.add(jLabel5, new XYConstraints(268, 11, 75, 21));
    contentPane.add(jProgressBar1, new XYConstraints(38, 154, 162, 17));
    contentPane.add(jButton1, new XYConstraints(49, 180, -1, 25));
    contentPane.add(jButton3, new XYConstraints(121, 180, -1, 24));
    contentPane.add(jRadioButton2, new XYConstraints(13, 129, 189, 21));
    contentPane.add(jRadioButton1, new XYConstraints(13, 103, 174, 21));
    contentPane.add(textArea1, new XYConstraints(269, 41, 191, 244));
    contentPane.add(jFileChooser1, new XYConstraints(314, 125, 62, 49));
    contentPane.add(jLabel4, new XYConstraints(123, 74, 24, 24));
    contentPane.add(jTxtFld2, new XYConstraints(151, 75, 49, -1));
    contentPane.add(jLabel6, new XYConstraints(6, 261, 254, -1));
    contentPane.add(jTxtFld4, new XYConstraints(35, 219, 169, -1));

  }
  /**Overridden so we can exit when window is closed*/
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JTextField jTxtFld1 = new JTextField();
  JLabel jLabel4 = new JLabel();
  JTextField jTxtFld2 = new JTextField();
  JLabel jLabel5 = new JLabel();
  JTextField jTxtFld3 = new JTextField();
  JButton jButton2 = new JButton();
  TextArea textArea1 = new TextArea();
  JFileChooser jFileChooser1 = new JFileChooser();
  JComboBox jComboBox1 = new JComboBox(new String[] {"IDEA" ,
                         "DES","DESede","Blowfish","GOST","CAST128","MARS"})
  JProgressBar jProgressBar1 = new JProgressBar();
  JLabel jLabel6 = new JLabel();
  JButton jButton3 = new JButton();
  JRadioButton jRadioButton1 = new JRadioButton();
  JRadioButton jRadioButton2 = new JRadioButton();
  JTextField jTxtFld4 = new JTextField();

  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
```

```
    }
  // 실행 버튼을 눌렀을 경우 수행되어져야 할 코드
    void jButton1_actionPerformed(ActionEvent e) {
        String algorithm = jComboBox1.getSelectedItem().toString();
                                                    FORM에서 주어진 알고리즘을 받음.
        Vector cipherValue , plainValue , printValue;
        int shiftTo , shiftFrom , value , minute, sec;
        int temp = 0;
        DataProcess cipherProcess , plainProcess;

        textArea1.setText("");
        textArea1.append("₩n₩n₩n" + "프로그램이 시작되었습니다." + "₩n");

// PENTIUM II 400M, 256M RAM 사양에서 실행했을 때 예상되는 실행시간을 표시하는 부분
        if(jRadioButton1.isSelected() == true){
            minute = (int)(available * 0.02)/60;
            sec = available % 60;
            textArea1.append("₩n₩n" + "예상시간은 " + minute + "분" + sec + "초입니다.");
            textArea1.append("₩n₩n₩n" + "암호화 KEY를 생성할시에는 ");
            textArea1.append("₩n₩n" + "3분 정도 더소요됩니다.");
        }else {
            minute = (int)(available * 0.0625);
            minute = (int)(minute * 0.02)/60;
            sec =  (int)((available * 0.0625) * 0.02);
            sec = sec % 60;
            textArea1.append("₩n₩n" + "예상시간은 " + minute + "분" + sec + "초입니다");
            textArea1.append("₩n₩n₩n" + "암호화 KEY를 생성할시에는 ");
            textArea1.append("₩n₩n" + "3분 정도 더소요됩니다.");
        }
        try{
            jTxtFld4.setText("Key생성중");
            CipherCreate cipher = new CipherCreate(algorithm);
                                                    // 주어진 알고리즘의 객체생성
            cipher.makeKey();  // 주어진 알고리즘의 KEY생성
            jTxtFld4.setText("암호문 생성중");
            cipher.makeCipher(fileName);  // 평문을 암호문으로 변환

            // shiftTo와 shiftFrom값을 매개변수로 넘겨줄것.
            shiftTo = Integer.parseInt(jTxtFld2.getText());  // SHIFT의 END 값
            shiftFrom = Integer.parseInt(jTxtFld1.getText()); // SHIFT의 START 값
            cipherProcess = new DataProcess(shiftFrom , shiftTo);
```

```
                plainProcess = new DataProcess(shiftFrom , shiftTo);

                if(jRadioButton1.isSelected() == true){
                                // SHIFT를 1BYTE 혹은 8BYTE중 어느 것으로 할지 결정
                    value = 1;
                }else {
                    value = 8;
                }
                // 암호화된 데이터를 정수형 값으로 VECTOR에 저장된 것을 넘겨 받음
                cipherValue = cipherProcess.makeData(new FileInputStream("cipher.txt") , value);
                //temp = cipherProcess.size();
                 // 평문 데이터를 정수형 값으로 VECTOR에 저장된 것을 넘겨 받음
                plainValue = plainProcess.makeData(new FileInputStream(fileName) , value);

                // 평문과 암호문을 SHIFT하여 결과를 넘겨 받음
                printValue = cipherProcess.processData(plainValue,cipherValue);
                String excelFileName = cipherProcess.printExcelFile(algorithm);
                String execValue = cipherProcess.printText();
                textArea1.setText("");
                textArea1.append(execValue);
                } catch(ArithmeticException exeption3){
                 // SHIFT의 범위를 잘못 지정했을 경우 SHIFT의 범위를 보여줌
                textArea1.setText("");
                textArea1.append("            NOTICE!! "+"₩n");
                textArea1.append("₩n");
                textArea1.append("          SHIFT값을"+"₩n"+"₩n");
    //          textArea1.append( "0 ~ "+ temp +" 이하로 입력하세요.!!");
    //          textArea1.append( "0 ~ "+ cipherValue.size() +" 이하로 입력하세요.!!");
            } catch(Exception exeption2){
                textArea1.setText("");
                textArea1.append("입력값에 오류가 있습니다.!!");
            }
        }
// FILE CHOOSER를 이용하여 평문파일을 선택하였을 경우 SHIFT의 가능 범위를 알려줌
    void jButton2_actionPerformed(ActionEvent e) {
      int i;
      byte [] value = new byte[1];
      jFileChooser1.showOpenDialog(jFileChooser1);
                                    // FILE CHOOSER를 실행 파일을 선택하게 함
      fileName = jFileChooser1.getSelectedFile().toString();
                                    // 선택된 파일의 이름을 넘겨 받음
```

```
        String str = fileName;
        jTxtFld3.setText(str.substring(str.lastIndexOf("₩₩")+1,str.length()))
        String inFile = str.substring(str.lastIndexOf("₩₩")+1,str.length());

        try {
            available = 0;
            FileInputStream inData = new FileInputStream(fileName);
            while(( i = inData.read(value)) != -1) {
                available++;
            }
        } catch(IOException e1){
        } catch(Exception e2) { }
       // FORM 화면 출력결과에 SHIFT 가능범위를 출력
        textArea1.setText("");
        textArea1.append("₩n₩n" + "1바이트 단위로 비교시");
        textArea1.append("₩n" + "  SHIFT의 범위입니다.");
        textArea1.append("₩n₩n" + " 0 ~ " + available);
        textArea1.append("₩n₩n₩n" + "8바이트 중 상위 1바이트비교");
        textArea1.append("₩n" + "  SHIFT의 범위입니다.");
        textArea1.append("₩n₩n" + " 0 ~ " + (int)(available*0.0625-10))
        i = (int)(available*0.0625-10);
        String temp = i + "";
        jTxtFld2.setText(temp);
      }
// 종료버튼을 눌렀을 경우 처리하는 FUNCTION
  void jButton3_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
// Radio 버튼 : 1BYTE단위로 SHIFT 하고자할 때 사용
  void jRadioButton1_actionPerformed(ActionEvent e) {
    jRadioButton2.setSelected(false);
    jRadioButton1.setSelected(true);
    String temp = available + "";
    jTxtFld2.setText(temp);

  }
// Radio 버튼 : 8BYTE중 상위 1BYTE를 SHIFT 하고자할 때 사용
  void jRadioButton2_actionPerformed(ActionEvent e) {
    jRadioButton1.setSelected(false);
    jRadioButton2.setSelected(true);
    int i = (int)(available*0.0625-10);
```

```
    String temp = i + "";
    jTxtFld2.setText(temp);
  }
}
```

(3) 각 암호화 알고리즘별로 객체를 만드는 클래스 정의

```
package cipherhigh;

import java.io.*;
import java.security.*;
import javax.crypto.*;

public class CipherCreate {

    private String cipherKey , algorithm;
    Key key;
    Cipher cipher;
// CONSTRUCTOR - 기본 생성자로 IDEA알고리즘으로 객체생성
    public CipherCreate() {

     cipherKey = "IDEA.key"; // 키 이름 설정
     algorithm = "IDEA";
     try{
         this.makePadding(); // 해당 알고리즘의 패딩 설정
     } catch (Exception e1){ }
     }
// CONSTRUCTOR - 주어진 알고리즘으로 객체 생성
    public CipherCreate( String cipherAlgorithm) {

     cipherKey = cipherAlgorithm + ".key"; // 키 이름 설정
     algorithm = cipherAlgorithm;
     try{
         this.makePadding(); // 해당 알고리즘의 패딩 설정
     } catch (Exception e2){ }
     }
// 키가 주어지지 않을 경우 키를 생성하고, 키가 주어지면 키를 생성하지 않는다.
    public void makeKey() throws Exception {

     try{
                    ObjectInputStream in = new ObjectInputStream(new
```

```
FileInputStream(cipherKey));
            key = (Key)in.readObject();
            in.close();
        } catch(Exception exception1) {
            KeyGenerator kg = KeyGenerator.getInstance(algorithm);
            kg.init(new SecureRandom());
            key = kg.generateKey();
                    ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(cipherKey));
            out.writeObject(key);
            out.close();
        }
        cipher.init(Cipher.ENCRYPT_MODE , key);
    }



// 기본 알고리즘(IDEA)의 패딩을 설정해주는 FUNCTION
    public void makePadding() throws Exception {
        String padd = algorithm + "/ECB/PKCS5Padding";
        cipher = Cipher.getInstance(padd);
    }
// 주어진 알고리즘으로 패딩을 설정해주는 FUNCTION
    public void makePadding( String cipherAlgorithm ) throws Exception {
        String padd = cipherAlgorithm + "/ECB/PKCS5Padding";
        cipher = Cipher.getInstance(padd);
    }
// 현재 주어진 알고리즘 이름을 반환하는 FUNCTION
    public String getAlgorithm() {
        return algorithm;
    }
// 알고리즘 설정 FUNCTION
    public void setAlgorithm( String cipherAlgorithm ) {
        algorithm = cipherAlgorithm;
    }
// 평문을 암호화 하는 FUNCTION
    public void makeCipher( String inFileName ) {
        try{
            FileInputStream in = new FileInputStream(inFileName);
            FileOutputStream out = new FileOutputStream("cipher.txt");
            CipherOutputStream cipherOut = new CipherOutputStream(out , cipher);
```

```
            byte [] buffer = new byte[8192];
            int lengths;

            while((lengths = in.read(buffer)) != -1) {
                cipherOut.write(buffer , 0 , lengths); // 평문이 암호화 됨
            }
            in.close();
            out.close();
            cipherOut.close();
        } catch (Exception exception2){ }
        }
}
```

(4) 평문 암호화된 데이터를 SHIFT하는 데이터 처리 클래스 정의

```
package cipherhigh;

import java.io.*;
import java.util.Vector;

public class DataProcess extends CipherHighByteSigmaFrame1{
    Vector dataValue ;
    Vector printValue;
    byte [] value;
    private int shiftTo , shiftFrom , max , min;
    long sum = 0;

// CONSTRUCTOR
    public DataProcess() {
     dataValue = new Vector();
     printValue = new Vector();
     shiftTo = 10;
     shiftFrom =  0;
    }
// CONSTRUCTOR
    public DataProcess(int start , int end) {
     this();
     shiftTo = end;
     shiftFrom =  start;
```

```
    }

    public Vector getDataValue() {
     return dataValue;
    }

    public Vector getPrintValue() {
     return printValue;
    }
// 원하는 SHIFT의 END 값을 얻는 FUNCTION
    public int getShiftTo(){
     return shiftTo;
    }
// 원하는 SHIFT의 END 값을 주어지는 FUNCTION
    public void setShiftTo(int start) {
     shiftTo = start;
    }
// SHIFT의 시작 값을 얻는 FUNCTION
    public int getShiftFrom(){
     return shiftFrom;
    }
// SHIFT의 시작 값을 주어지는 FUNCTION
    public void setShiftFrom(int end) {
     shiftFrom = end;
    }
// SHIFT하기 위해 주어진 파일의 데이터를 정수형 값으로 변환하여 반환하는 FUNCTION
    public Vector makeData( FileInputStream data , int a) {
     int i = 0;
     int temp;

        if( a == 1) {
            value = new byte[1]; // 주어진 데이터를 1BYTE 단위로 정수형 값으로 바꿈
        } else {
            value = new byte[8]; // 주어진 데이터를 8BYTE 단위 중 상위 1BYTE를 정
수형으로 바꿈
        }

    try{
        while((i = data.read(value)) != -1) { // FILE에서 데이터를 읽어옴
            data.read(value);
            String str = String.valueOf(value[0]);
```

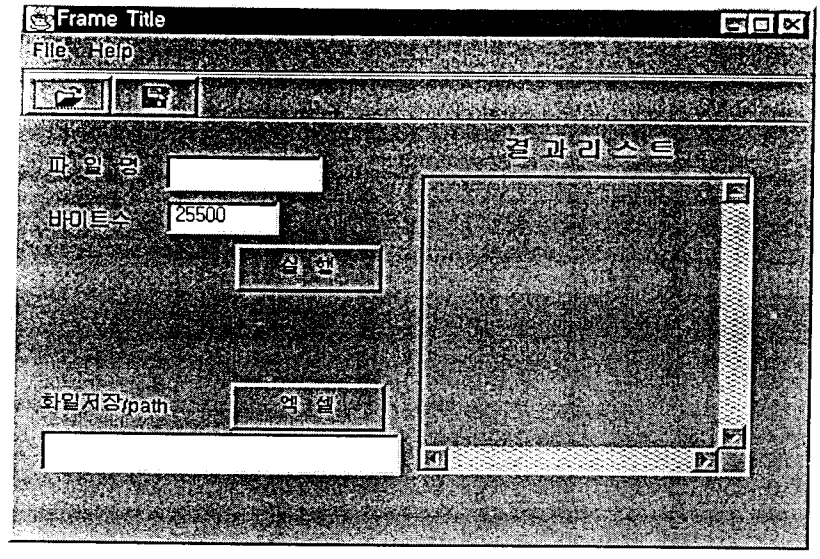
```
                temp = Integer.parseInt(str);  // 정수형으로 변환

                if(temp < 0) {
                    temp += 256;
                }

                String strTemp = String.valueOf(temp);
                dataValue.addElement(strTemp); VECTOR에 저장
            }
        } catch(IOException e) { }

            return dataValue;
        }
    // 주어진 두 개의 데이터(평문, 암호문)를 서로 SHIFT하는 FUNCTION
        public Vector processData( Vector data1 , Vector data2 ) {
        int i , j , k;

        for( i = 0 ,  k = shiftFrom ; k <= shiftTo ; k++ , i++){
            for( j = 0 ; j < data2.size() - (k + 1) ; j++){
                // VECTOR에 저장되어진 데이터를 정수형으로 형 변환하여 모두 더한다
                long dataValue1 = Long.parseLong((String)data1.elementAt(j));
                long dataValue2 = Long.parseLong((String)data2.elementAt(j + k));
                sum = sum + dataValue1 * dataValue2;
            }

            String strTemp = String.valueOf(sum / (data1.size() - k));
            printValue.addElement(strTemp);
            sum = 0;
                if((k%100) == 0){
                    jTxtFld4.setText("SHIFT : " + k + " 번 중");
                }
        }

        return printValue;
        }

    // SHIFT한 결과를 FORM화면 출력결과에 간단하게 보여주는 FUNCTION
        public String printText() {
        int i , k;
        String printData = "";
```

```
        for( k = 0 , i = shiftFrom ; i <= shiftTo ; k++ , i++){

//          printData += "SHIFT " + i + " : " printValue.elementAt(k) + "₩n";
           // VECTOR에 저장되어 있는 데이터를 정수형으로 형 변환
            long printValueMax = Long.parseLong((String)printValue.elementAt(max));
            long printValueMin =  Long.parseLong((String)printValue.elementAt(min));
            long printValueNew = Long.parseLong((String)printValue.elementAt(k));

            if( printValueMax < printValueNew) max = k; //SHIFT한 것 중에서 가장 큰 값
저장
            if( printValueMin > printValueNew) min = k;  //SHIFT한 것 중에서 가장 작은 값
저장
        }
        // FORM화면 출력결과에 데이터처리 결과를 보여줌
        printData += "₩n₩n" + "## Maximum Value ## ₩n";
        printData += "SHIFT ";
        printData += (max + shiftFrom) + " : " + printValue.elementAt(max);
        printData += "₩n₩n" + "## Minimum Value ## ₩n";
        printData += "SHIFT ";
        printData += (min + shiftFrom) + " : " + printValue.elementAt(min);
        return printData;
       }
// 각각SHIFT한 결과를 엑셀 파일로 저장하는 FUNCTION
       public String printExcelFile(String str) throws IOException {
        int i , k;
        String fileName = str + ".xls";  // 엑셀 파일 이름 지정

        BufferedWriter out = new BufferedWriter(new FileWriter(fileName));

        for( k = 0 , i = shiftFrom ; i <= shiftTo ; k++, i++){
            long printValueElement = Long.parseLong((String)printValue.elementAt(k));
            out.write("SHIFT" + "₩t" + i + "₩t" + printValueElement);
            out.newLine();
        }
             out.close();
             return fileName;
       }
// 데이터의 SIZE를 반환하는 FUNCTION
/*     public int size() {
          return dataValue.size();
       } */
```

}

나. ASCII의 빈도수를 구하는 프로그램

(1) FORM의 메인 화면



[그림 4-4]

(2) 소스 코드 및 코드 설명

다음 코드는 어느 지정한 파일을 선택적으로 필요한 바이트만큼 읽어들여 빈도수와 확률을 출력하는 소스이다. 자바개발툴인 제이빌더에서 작성한 코드의 일부분이다.

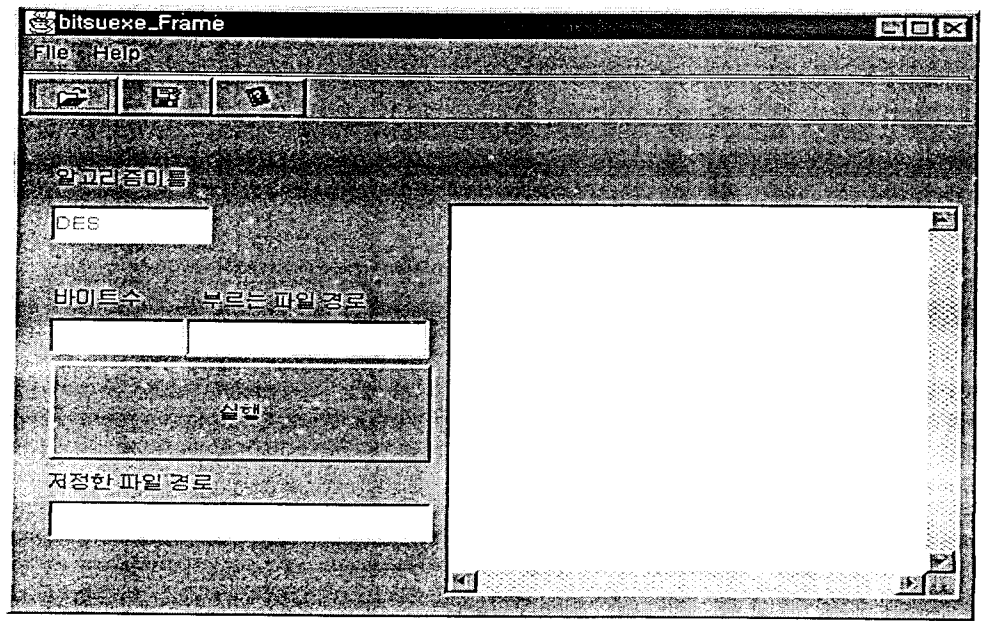
```
void butt1_actionPerformed(ActionEvent e){
        int variable1=0;                          // 변수들을 정의한다.
        txta.setText(null);
        int bytesu;
        int i; int n; int j; int w;
        float temp1=0, temp2=0;
        byte k=-128;
      float[][] test= new float[256][3];          // 값을 저장할 배열을 선언
      int total_byte =0 ;
        bytesu =Integer.parseInt(txtf2.getText());
      for( j=0 ; j<256; j++){
            test[j][0]=k++;
     }
     try{
        DataInputStream in = new DataInputStream(new FileInputStream(openfile));
        BufferedWriter out = new BufferedWriter(new FileWriter("imsiFile"));
        txtf4.setText("imsiFile");
```

```
if(bytesu == 0){                          // 0을 입력하면 화일을 모두 변환
  while((i=in.read()) != -1) {
        k=(byte)i;
           for(n=-128,j=0 ; (k != n) && (n < 128) ; n += 1, j++)    ;
             test[j][1] += 1;
             total_byte++;
   }
 }
else{                                     // 바이트수 만큼 화일을 아스키로 변환
  for(w=0; w< bytesu; w++){
      i=in.read();
      k=(byte)i;
        for(n=-128,j=0;(k != n)&&(n<128);n+=1,j++)    ;
          test[j][1]+=1;
          total_byte++;
           if( total_byte != 0){
             temp1 = (float)total_byte;
             test[j][2] = test[j][1] /temp1;
           }
      }
}
for(i = 0; i<256 ; i ++) {
  if(test[i][1] != 0) {
   test[i][2]= (test[i][1] / total_byte);
   }
 else {
    test[i][2] = 0;
   }
}
for(i=128; i<256 ; I++){          // 빈도수와 백분율을 구함
      temp1 = test[i][2];
      out.write((int)test[i][0] + "₩t" + (int)test[i][1] + "₩t" + temp1 + "%")
      out.newLine();
   }
  for(i=0; i<128 ; i++){
      test[i][0] = test[i][0] + 256;
      temp2 = test[i][2];
      out.write((int)test[i][0] + "₩t" + (int)test[i][1]+"₩t" +  temp2 +"%");
      out.newLine();
   }
  out.newLine();
```

```
            out.write("₩t"+ total_byte+"₩t");
            out.close();
            FileReader fr= new FileReader("imsiFile");
            BufferedReader br=new BufferedReader(fr);
            String readLine;
            txta.append("숫자"+"          "+"빈도수"+"          "+"퍼센트"+"₩n");
            for(;(readLine=br.readLine()) !=null; ){
                txta.append(readLine + "₩n");
                  variable1=variable1+1;
             }
            br.close();
            }
            catch(IOException e1){
             System.out.println("reading/writing error" + e1);
             return;
        }
    }
```

다. 암호화된 파일의 ASCII 코드 값의 빈도수를 구하는 프로그램

(1) FORM의 메인 화면



[그림 4-5] 암호화된 파일의 ASCII 코드 값의 빈도수를 구하는 프로그램 메인화면

(2) 소스 코드 및 코드 설명

다음은 위와 비슷한 스타일의 코드인데 이코드의 목적은 대칭키알고리즘으로 나온 문서의 원하는 바이트만큼의 아스키빈도수를 출력하는 소스의 주요부분이다.

```
    void exebut_actionPerformed(ActionEvent e) throws Exception{
      int KeyBufferSize = 8192;                    // 필요한 변수를 선언
      String ciph = algoltxt.getText().toString();
      int value = 0;
      int bytesu = 0;
      int variable1=0;
      int i=0, w=0, j=0, n=0;
      Key key;
      float[][] test = new float[256][3];          // 값을 저장하기 위한 배열선언
      int total_byte = 0;
      float temp1 = 0, temp2=0;
      showdatatxta.setText(null);
      byte k = -128;
      bytesu = Integer.parseInt(bytesutxt.getText());
      for ( j = 0 ; j < 256 ; j++) {
        test[j][0] = k ++;
      }
      // 파일을 읽기 전에 대칭키의 유무를 파악하고 없으면 생성, 있으면 사용하는 부분

     try {
      ObjectInputStream in = new ObjectInputStream(new FileInputStream("Secretkey."+ ciph))
      key = (Key)in.readObject();
      in.close();
    }catch(Exception e1){
      KeyGenerator kg = KeyGenerator.getInstance(ciph);
      kg.init(new SecureRandom());
      key = kg.generateKey();
      ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("Secretkey."+ciph));
      out.writeObject(key);
      out.close();
    }
  Cipher cipher = Cipher.getInstance(ciph + "/ECB/PKCS5Padding");
  cipher.init(Cipher.ENCRYPT_MODE, key);

// 암호화할 파일을 읽어서 암호화, 이후에 암호화한 파일을 읽어서 빈도수와 백분율을 구함.
  try {
      FileInputStream in = new FileInputStream(openfile);
      FileOutputStream out = new FileOutputStream("imsifile.txt");
      CipherOutputStream CipherOut = new CipherOutputStream(out, cipher);
      byte[] buffer = new byte[KeyBufferSize];
      int lengths;
```

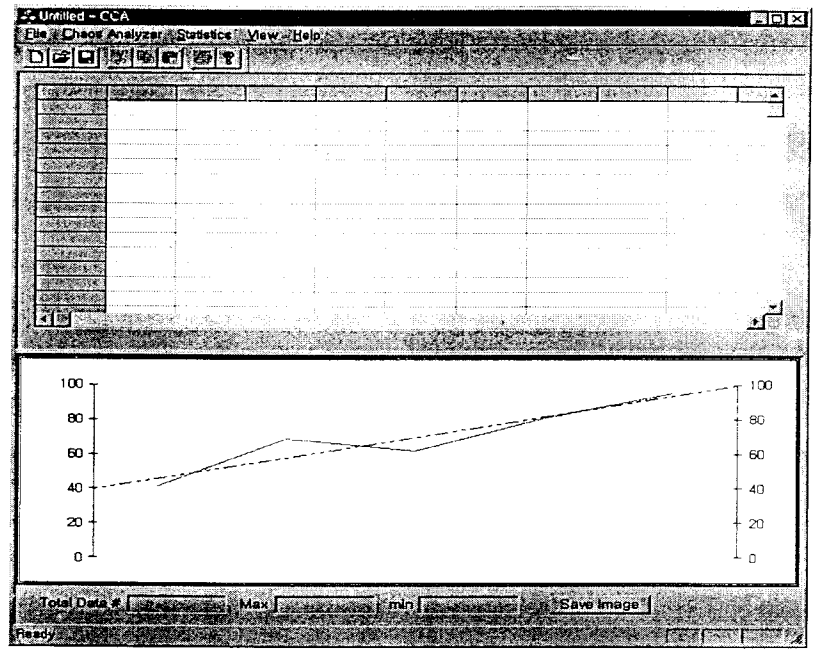
```
    while((lengths = in.read(buffer)) != -1)
      CipherOut.write(buffer,0,lengths);
    in.close();
    CipherOut.close();
  }catch(Exception e3){};
  .....
```

4. 카오스 분석기의 개발

(Cryptography-Chaos Analyzer : CAA)

가. CCA(Cryptography-Chaos Analyzer) 시스템

본 CCA(Cryptography-Chaos Analyzer) 프로그램은 암호화 알고리즘의 비선형성을 측정하기 위해 개발되었다. 본 시스템은 입력받은 일차원 시계열 데이터에 대해 확정적 혼돈을 규명하는 시스템으로 개발되어야 한다. 특히, 윈도우즈 프로그램에서의 단점중 하나인 DC(Device Context)를 효율적으로 사용하기 위해 비쥬얼 C++ 5.0에서 제공하는 FlexGrid와 비쥬얼 베이직 5.0에서 지원하는 Chart 콘트롤(.ocx)을 사용하였다. 다음은 개발된 시스템의 초기화면이다.

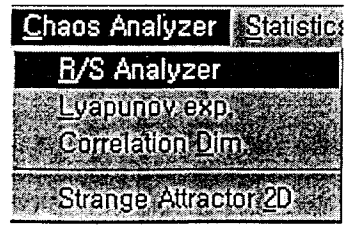


[그림 4-6] CCA 시스템

앞의 그림에서 위쪽 윈도우는 데이터와 처리된 결과를 표시하는 창이며 아래는 이를 그래프로 나타낸 주는 창이다. 아래 그래프에서 검정 실선으로 나타나는 것은 추세선을 의미하며, 적색선은 데이터를 나타낸다.

나. 시스템 기능

본 시스템의 주된 기능은 메뉴바에 있는 Chaos Analyzer 메뉴에 있다. 다음 그림은 Chaos Analyzer 메뉴를 구성하고 있는 아이템이다.



[그림 4-7] Chaos Analyzer 메뉴 구성

(1) R/S 분석

$$(R/S)_n = C \cdot n^H$$
$$where,\ H\ is\ Hurst\ exponent$$
$$R/S\ is\ ReScaled\ range$$
----------(식 28)

재구성된 범위는 평균값이 0이며, 국부적 표준편차를 의미하며 양변에 log를 취하면 다음과 같이 된다.

$$\log(R/S_n) = H\log(n) + \log(c)$$
----------(식 1)

R/S는 시간의 제곱근의 증가만큼으로 빠르게 증가하며 다음과 같은 과정을 통하여 구해진다.

① 길이 M의 시계열인 경우, 이를 길이 N = M -1을 가지는 로그 비를 가지는 값으로 변환한다.

$$N_i = \log(M_{(i+1)} / M_i)$$
$$where,\ i=1,2,\cdots,(M-1)$$
----------(식 2)

② $A \cdot n = N$ 이 되도록 길이 n을 가지는 부주기(subperiod) A를 구하고 각각의 부주기에 레이블을 $I_a\,(a=1,2,\cdots,A)$을 붙이고, I_a의 각각의 원소들에 대해 $N_{k,a}(k=1,2,\cdots,n)$인 값을 정의한다. 길이 n의 I_a에 대하여 평균값은 다음과 같이 정의된다.

$$e_a = (\frac{1}{n}) \sum_{k=1}^{n} N_{k,a}$$
----------(식 3)

다시 말하면, e_a는 길이 n을 가지는 I_a를 포함하는 N_i의 평균값이다.

③ 각각의 I_a에 의한 평균값으로부터, 누적변경(Accumulated departures)의 시계열 $(X_{k,a})$는 다음과 같이 정의된다.

$$X_{k,a} = \sum_{i=1}^{k}(N_{i,a} - e_a) \qquad where,\ k=1,2,\cdots,n$$
----------(식 4)

④ 범위는 부주기 I_a 사이의 $X_{k,a}$의 최대값과 최소값의 차로 얻어진다.

$$R_{I_a} = Max(X_{k,a}) - \min(X_{k,a})$$
$$where,\ k=1,2,\cdots,n$$
----------(식 5)

⑤ 각각의 I_a에 대한 표준편차는

$$S_{I_a} = \sqrt{(\frac{1}{n}) \cdot \sum_{k=1}^{n}(N_{k,a} - e_a^2)}$$
----------(식 6)

이다.

⑥ 각 범위 R_{I_a}는 S_{I_a}의 비로써 정규화 되어 진다. 그러므로 재구성된 범위(R/S)는 각각의 I_a에 대한 R_{I_a}/S_{I_a}와 같다. 위의 과정 2로부터 우리는 길이 n의 연속되는 값 A를 얻었다. 그러므로,

$$(R/S)_n = (\frac{1}{A}) \cdot \sum_{a=1}^{A} (\frac{R_{I_a}}{S_{I_a}}) \quad \text{----------(식 7)}$$

로 정의되어 진다.

⑦ 길이 n은 다음 높은 값으로 증가되고, $(M-1)/n$은 정수값을 가진다. $n = (M-1)/2$ 까지 과정 1부터 6까지를 반복함으로써 우리는 시계열의 초기점과 종말점에 n을 사용할 수 있다. $\log(n)$을 독립변수로, $\log(R/S_a)$을 종속변수로 사용함으로써 최소 자승 회귀(least squared regression)가 이루어지며, 여기서 얻은 값의 기울기로써 허스트 지수 H가 구해진다.

(2) Lyapunov 지수

비선형 동태 방정식을 가지는 시스템처럼 랜덤한 요소를 가지고 있는 경우에는 어떤 시점에서 발생한 충격(shock)이 유한한 시간이 지난 후에 영향력을 모두 잃어버리게 된다. 즉, 시스템의 운동방향은 충격에 따라 어떠한 방향으로 진행되게 되고 소멸되어진다. 이런 시스템의 확장 또는 수축을 의미하는 것이 리아프노프 계수(Lyapunov exponent)이다. 일반적으로 i 차원의 $(p_i(t))$에서 리아프노프 계수 (L_i)를 측정하기 위한 방법은 다음과 같다.

$$L_i = \lim_{t \to \infty} (\frac{1}{t}) \log_2 (\frac{p_i(t)}{p_i(0)})$$

로 측정할 수 있다. 실제적으로 리아프노프 발산 지수를 측정하기 위하여 먼저 알아야 할 자료들에는 입력자료, 매립 차원(Embedding Dimension), 시간 지연(Time Lag), 반복 횟수(Evolution Time), 최대/최소 거리(Max/min Distance)와 평균 오비탈 주기(Mean Orbital Period)를 알아야 한다. 다음은 리아프 노프 계수 측정 알고리즘을 나타낸 것이다.

- Embedding Dim(m) : Correlation integral Cm(R)
- Time Lag : m * t(time lag) = Q(period)
- Mean Orbital Period : R/S analysis(residual analysis를 전처리 과정으로 수행하는 것이 좋다)
- Max/min Distance : 임베딩 디멘전을 이용하여 최소값과 최대값을 정한다.

① 적어도 하나의 평균 오비탈 주기를 떨어진 두 점을 선택하여 두점간의 거리를 측정한다 $(DI) - L_1(x_0)$

② 리아프노프 지수 측정간격을 결정한다. $(DF) - L_1(x_1)$

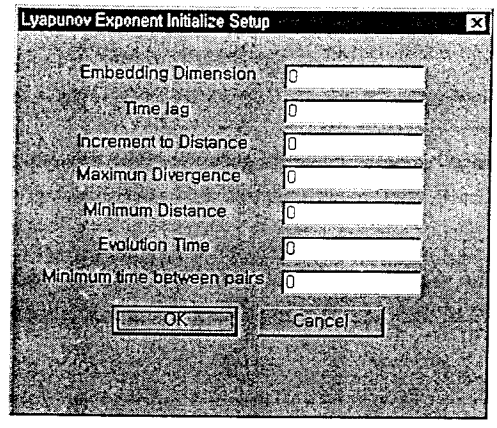
③ 리아프노프 지수를 초기화한다.

$$SUM = \frac{1}{t}\sum_{j=1}^{m}\log_2\left(\frac{L(x_{j+1})}{L(x_j)}\right) + SUM \quad \text{----------(식 8)}$$

$$zLyap = \frac{SUM}{Iteration} \quad \text{----------(식 9)}$$

④ 만약 DF > Dist_Max이고 DF < Dist_min이면 새로운 점을 선택한다.(DN) 단, 시간간격은 적어도 한 주기 이상이 되어야 하며, 최소/최대점간의 거리 내에서 떨어져 있어야 하며 위상공간에서 이루는 DN과 DF간의 각도가 최소가 되는 점을 선택한다. 그리고 DI = DN으로 설정하고 3)의 과정부터 재 시작한다.

다음의 그림은 리아프노프 지수 측정을 위해 초기값을 지정하기 위한 대화상자이다.



[그림 4-8] 대화상자

(3) 상관 차원 (Correlation Dimension)의 측정

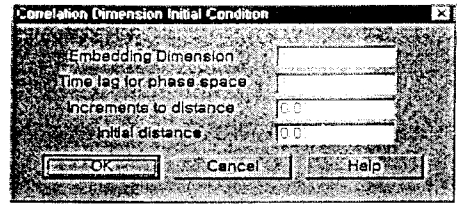
Grassberger와 Proccacia에 의해 제시되어진 상관차원은 시계열 자체가 하나의 변수에 의해 관찰되어지므로 1에서 2사이의 값밖에 가지지 못하는 한계를 가지고 있다. 그러나 위상공간에서 시계열을 재구성하는 경우에는 모든 독립변수들을 관찰할 수 있으므로 시계열을 위상공간에 재구성하는 것이 일반적이다. 이에 근거하여 제시된 상관 차원은 매립차원 m=2부터 증가 시면서 프랙탈 구조의 반경(diameter)을 증가하여 측정한다.

$$C_m(R) = \left(\frac{1}{N^2}\right) * \sum_{i,j=1}^{N} Z(R - |x_{i-x_j}|)$$

$$Where, Z(x) = 1 \ if \ R - |x_i - x_j| > 0 \ ; \ 0 \ otherwise$$
$$N = Number\ of\ observations$$
$$R = distance$$
$$C_m = correlation\ integral\ for\ dimension\ m$$

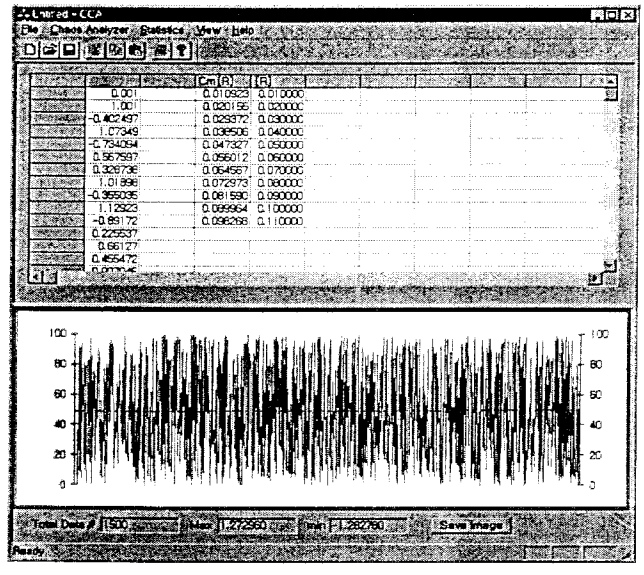
여기서 $Z(x)$는 Heaviside Function이라고 불리우며, 상관차원은 t 와 $t+1$점간에서 프랙탈 반경 안에 두 점이 있을 확률을 의미한다.

다음의 그림은 상관차원 측정 초기값을 지정하기 위한 대화상자이다.



[그림 4-9] 초기값 지정 대화상자

초기값을 지정한 후 상관차원을 측정하면 다음과 같은 결과가 나타난다.



[그림 4-10] 상관차원 측정값

위에서 나온 결과를 바탕으로 하여 $\log(Cm[R])$과 $\log([R])$의 그래프를 그려 기울기를 구하면 상관차원을 구할 수 있다.

(4) 끌개(Attractor) 구성

어떤 시계열 패턴(pattern)이 샘플링(sampling) 간격 δt로 n개의 데이터를 가지고 있다고 하면 시계열은 다음과 같이 표현된다.

$$\{X(t), X(t+\delta t), X(t+2\cdot\delta t), X(t+3\cdot\delta t), \cdots, X(t+(n-1)\cdot\delta t)\}$$

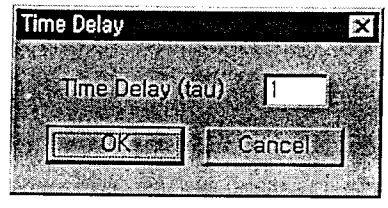
이때 시간지연(T)을 샘플링시간 δt의 2배로 잡고, 3차원 벡터열을 구하면 다음과 같다. ($T = 2\,\delta t$)

$$\{X(t), X(t+2\cdot\delta t), X(t+4\cdot\delta t)\}, \{X(t+1\cdot\delta t), X(t+3\cdot\delta t), X(t+5\cdot\delta t)\}, \cdots,$$

$$\{X(t+(n-5)\cdot\delta t), X(t+(n-3)\cdot\delta t), X(t+(n-1)\cdot\delta t)\}$$

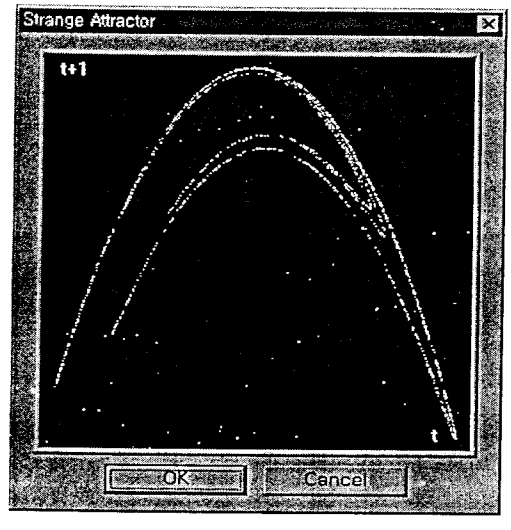
이렇게 정해진 점들을 3차원공간에 찍으면 원래 계의 운동 성질을 보이는 3차원끌개를 얻게된다. n값이 끌개의 원래 차원 값보다 같거나 큰 범위에서 잘 선택되면 이 벡터 열은 원래의 운동과 동일한 성질을 보이게 된다.

이와 같은 방법으로 재구성된 기이한 끌개는 물론 원래의 끌개와 정확히 같은 형태는 아니고, 일반적으로 변형된 형태를 갖는다. 하지만 발산(Lyapunov) 지수와 프랙탈 차원은 이러한 변형에 대해 변하지 않기 때문에 재구성된 끌개로부터 이들 값을 계산해 낼 수 있다. 본 시스템에서는 2차원 끌개를 구성하는 것을 목표로 하였다. 메뉴를 선택하면 끌개를 구성에 필요한 시간지연 설정 대화상자가 나타난다.



[그림 4-11] 시간 지연 대화상자

다음의 그림은 비선형 끌개를 구성하는 것으로 잘 알려진 Henon Map 데이터를 가지고 시간지연 1의 값을 지정하여 계산된 그림이다.



[그림 4-12] Henon Map

5. Packet Caturing

가. VPACKET

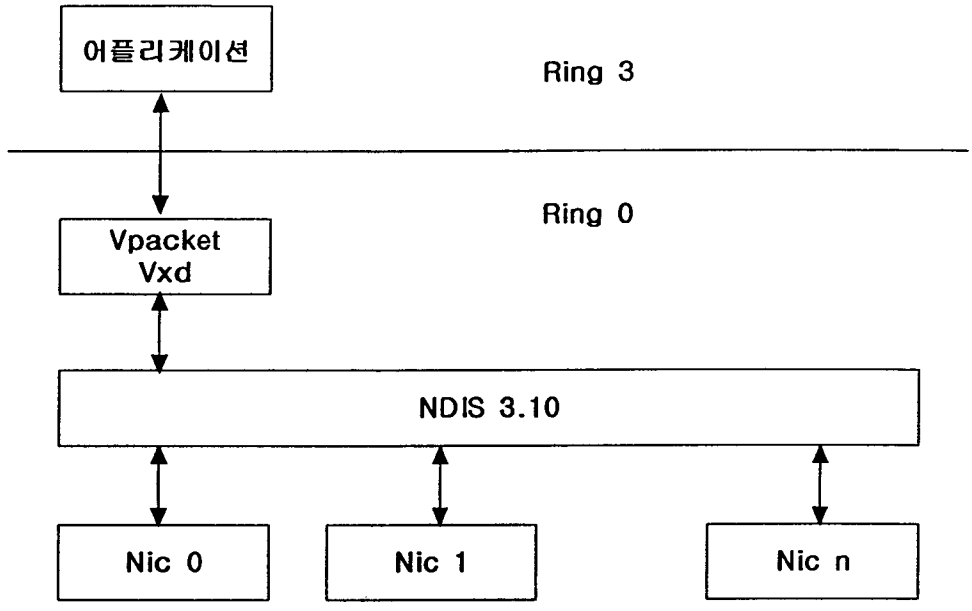
VPACKET이란 PC에 설치되어 있는 어떤 Network Interface Controller (NIC) 에 WIN32 어플리케이션이 직접 접근할 수 있도록 만든 윈도우 95/98/me의 가상 장치 드라이버이다.

Direct Network Access는 네트워크 모니터링 어플리케이션을 개발하거나 프로그래머들이 자신만의 프로토콜 스택을 구현하고자 할 때에 매우 유용하게 사용할 수 있다. VPACKET 드라이버는 P32 Programming Environment[1] 의 일부분이다. P32 Programming Environment는 WIN32 패키지로, Canberra 대학에서 운영체계와 프로토콜 설계 개념을 가르치는 데 사용하고

있는 툴이다.

(1) 구 성

WIN32 프로그래밍 환경은 하위 레벨의 네트웍 접근을 할 수 있는 직접적인 방법을 제공하지는 않는다. 따라서 하위 레벨의 네트웍 접근을 해야 하는 어플리케이션은 일반적으로 VxD라고 부르는 Virtual Device Driver를 사용하고 있는데, 이 VxD는 하부의 NIC과 상부의 WIN32 어플리케이션 사이의 인터페이스 역할을 하고 있다. [그림 4-] 참조



[그림 4-13] 네트워크 구조

어플리케이션은 먼저 WIN32의 CreateFile 함수를 이용하여 VxD를 적재해야 한다. 그리고 VxD가 제공하는 서비스들을 이용하기 위해서 WIN32의 DeviceIoControl 함수를 호출할 수 있다.

(2) Interface Abstraction Layer

[그림 4-4]에서 본 것처럼 VxD는 설치되어 있는 NICs에 직접 접근하는 것이 아니다. NDIS3.10이라고 불리는 인터페이스 추상 층 (Interface Abstraction Layer)이 네트워크의 하드웨어와 VxD 사이에 존재한다. 이 인터페이스 추상 층은 NIC에 접근하려는 소프트웨어가 하부의 하드웨어 어댑터의 특정부분에 접근하는 것을 방지해준다. 따라서 VPACKET VxD는 NDIS를 지원하는 어댑터를 제공하는 기계에 설치된 어떤 NIC와도 교신할 수 있다.

NDIS호환 정도는 어댑터마다 다르다는 것을 염두해 두자. 특히 마이크로소프트의 다이얼 업 어댑터(PPPMAC)은 NDIS에서 지원하지 않는다. 그 이유는 NdisSend함수가 PPP Link를 통해서 데이터를 보낼 수 없기 때문이다.

(3) VxD적재하기

WIN32 어플리케이션은 WIN32의 CreateFile함수를 호출하면서 특별한 Naming Convention을 이용하여 VxD를 메모리에 적재한다. 아래의 소스코드는VPACKET VxD를 어떻게 적재하는

지 보여준다.

```
#include <windows.h>

HANDLE hVxD;

hVxD = CreateFile("WWWW.WWVPACKET.VXD",
                  GENERIC_READ | GENERIC_WRITE,
                  0,
                  NULL,
                  OPEN_EXISTING,
                  FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED |
                  FILE_FLAG_DELETE_ON_CLOSE,
                  NULL);

if (hVxD == INVALID_HANDLE_VALUE)
   return SYSERR;
```

첫번째 인자는 VPACKET.VXD를 메모리에 적재한다는 것을 가리키고 있고, 여섯번째 인자는 드라이버가 asynchronous 오퍼레이션 (FILE_FLAG_OVERLAPPED)을 지원하고 VxD가 종료될 때 메모리에서 제거한다는 것을 지정하고 있다 (FILE_FLAG_DELETE_ON_CLOSE).

asynchronous 오퍼레이션을 요청하는 함수는 즉시 함수를 리턴한다. 즉 요청한 오퍼레이션이 처리될 때 가지 기다리지 않는다. 어플리케이션이 오퍼레이션이 처리되었는지를 확인하려면 다른 방법을 사용해야 한다.

CreateFile에서 리턴받은 핸들은 일반적인 파일 핸들이 아니라 어플리케이션에서 나중에 Device Driver의 서비스에 접근할 때 사용하기 위한 핸들이다.

VPACKET VxD 는 여러 번 열 수 있다. CreateFile을 호출 할 때마다 각각 다른 핸들을 리턴하는 데 처음 호출 할 때는 실재로 드라이버를 적재하고 실행하지만 다음 호출부터는 단지 새로운 핸들만을 만들어서 리턴한다.

VPACKET VxD는 사전에 설치하거나 Setup을 해야 하는 일을 필요로 하지 않는다. 즉 이 드라이버에 대한 어떤 INF 파일도 존재하지 않고 단지 모든 설치는 드라이버가 NIC에 바인딩되는 런 타임에 이루어진다.

(4) VxD 제거하기

VxD는 WIN32 CloseHandle함수를 호출하여 종료시킨다. 이 때 인자로 CreateFile에서 리턴된 핸들을 넘겨주어야 한다. 만약 디바이스 드라이버가 여러 번 열린 상태라면 , VxD는 모든 핸들이 닫힌 경우에 메모리에서 제거된다.

Network Interface와 바인딩하기

일단 메모리에 적재되고 실행이 되면 VPACKET VxD는 특정 NIC와 바인딩해야 한다. 바인딩은 아래의 Bind함수를 호출함으로서 이루어진다.

```
int Bind(HANDLE hVxD, BYTE* inBuffer)
{
    HANDLE          hEvent;
    DWORD           cbRet;
    OVERLAPPED      ovlp    = {0,0,0,0,0};

    int result;
    int cbIn = 5;

    hEvent = CreateEvent(0, TRUE, 0, NULL);
    if (!hEvent)
       return SYSERR;

    ovlp.hEvent = hEvent;

    result = DeviceIoControl(hVxD,
                             IOCTL_PROTOCOL_BIND,
                             inBuffer,
                             cbIn,
                             inBuffer,
                             cbIn,
                             &cbRet,
                             &ovlp);
    if (!result)
       GetOverlappedResult(hVxD, &ovlp, &cbRet, TRUE);
    CloseHandle(hEvent);
  return OK;
}
```

첫번째 인자는 CreateFile에서 얻은 핸들이고 두 번째 인자는 이 핸들과 바인딩하고자 하는 어댑터의 문자열 이름값이다. 이 지정된 문자열은 윈도우95의 아래 레지스트리를 살펴보면 확인할 수 있다.

HKEY_LOCAL_MACHINE₩System₩CurrentControlSet₩Services₩Class₩Net

일반적으로 다이얼 업 어댑터는 "0000"이고 이더넷 어댑터는 "0001"과 같이 이름이 부여된다. 그러나 어플리케이션은 원하는 어댑터와 바인드 하려면 레지스트리를 찾아서 어댑터 이름을 확인해야 할 필요가 있다.

어떤 다른 작업을 수행하기 전에 드라이버 핸들은 반드시 어댑터와 연결이 되어있는 것이어야 한다.

(5) 디바이스 드라이버 API

WIN32어플리케이션은 디바이스 드라이버의 서비스를 사용하기 위해서 DeviceIoControl 함수를 사용해야한다. 두 번째 인자는 아래와 같은 함수 코드중에 하나를 선택해야 해서 얻고자 하는 서비스를 선택할 수 있다.

IOCTL_PROTOCOL_QUERY_OID	Object ID 얻어내기
IOCTL_PROTOCOL_SET_OID	Object ID 설정하기
IOCTL_PROTOCOL_STATISTICS	Adapter statistic 알아내기
IOCTL_PROTOCOL_RESET	어댑터 초기화
IOCTL_PROTOCOL_READ	Packet 받기
IOCTL_PROTOCOL_WRITE	Packet 보내기
IOCTL_PROTOCOL_MACNAME	드라이버 이름 알아내기
IOCTL_PROTOCOL_BIND	어댑터와 VPACKET 바인딩

(6) Asynchronous 오퍼레이션

아래는 WIN32에서 Asunchronous 오퍼레이션을 어떻게 처리하는 방법이다.

Bind함수를 살펴보면 OVERLAPPED구조체의 멤버 hEvent는 CreateEvent함수에서 리턴된 값으로 설정한다. 그리고 나머지 멤버는 0으로 설정한다. 그리고 OVERLAPPED구조체의 주소값을 DevIoControl을 호출 할 때 마지막 인자값으로 넘겨준다. 디바이스 드라이버는 오퍼레이션을 시작하고 리턴한다. 요청된 오퍼레이션이 완료되면 드라이버에 의해서 특정 이벤트가 발생된다. 이 때 WIN32어플리케이션에서는 다른 임무를 수행할 수 있다. Bind함수의 경우, 바인드가 완료될 때 까지 아무일도 할 수 없으므로 GetOberlappedResult 함수를 호출하고 이벤트가 발생될 때 까지 블러킹하고 기다린다.

Asynchronous I/O메카니즘의 장점은 네트워크 어댑터를 통해서 받은 데이터를 읽어올 때 뚜렷이 확인 할 수 있다. 어플리케이션은 언제 패킷이 네트워크에 도달할 지 알 수 없으므로 다른 일을 처리하고 있다가 GetOverlappedResult을 마지막인자로 FALSE를 지정하고 호출하여 데이터가 도달했는지 확인 할 수 있다. 만약에 GetOverlappedResult함수가 FALSE를 리턴하고 GetLastError결과 ERROR_IO_PENDING을 리턴한다면 어플리케이션은 어떤 데이터도 도달하지 않았다고 판단하게 된다. 만약 GetOverlappedResult 함수고 TRUE를 리턴한다면 어플리케이션은 데이터가 도착했다고 판단하고 데이터를 처리한다.

Asynchronous I/O를 처리하는 다른 방법으로는 Synchronous 처리를 다른 쓰레드로 독립시켜 실행시키고 Global이벤트를 발생시켜서 메인 어플리케이션에서 그 이벤트를 기다리도록 만드는 방법을 고려해볼 수 있다.

VPACKET 가상 장치 드라이버는 간단하고 강력한 메커니즘으로 윈도우95에서 실행되는 WIN32 어플리케이션에 NIC에 직접 접근 할 수 있는 기능을 제공해준다.

(7) 추가 코드

IOCTL_PROTOCOL_QUERY_OID

이 오퍼레이션은 특정 Object ID를 리턴하는 예 이다.

```
BYTE iBuf[sizeof(PACKET_OID_DATA) + 128];
PPACKET_OID_DATA pOidData = (PPACKET_OID_DATA) iBuf;
int  result;
memset(iBuf, 0, sizeof(iBuf));
pOidData->Oid     = OID_GEN_CURRENT_PACKET_FILTER;
pOidData->Length  = 4;
result = ControlPacket((HANDLE) etptr->handle,
                       IOCTL_PROTOCOL_QUERY_OID,
                       iBuf,
                       sizeof(PACKET_OID_DATA) + 4,
                       iBuf,
                       sizeof(PACKET_OID_DATA) + 4);
if (result == 12) {
    memcpy(arg1, pOidData->Data, 4);      /* arg1 is an int * */
    return OK;
}
return SYSERR;
```

IOCTL_PROTOCOL_SET_OID

이 오퍼레이션은 특정 Object ID을 설정해준다.

```
pOidData->Oid     = OID_GEN_CURRENT_PACKET_FILTER;
pOidData->Length  = 4;
pOidData->Data[0] = (UCHAR) NDIS_PACKET_TYPE_PROMISCUOUS;

result = ControlPacket((HANDLE) etptr->handle,
                       IOCTL_PROTOCOL_SET_OID,
                       iBuf,
                       sizeof(PACKET_OID_DATA) + 4,
                       iBuf,
                       sizeof(PACKET_OID_DATA) + 4);
if (result > 0)
    return OK;

return SYSERR;
```

IOCTL_PROTOCOL_STATISTICS

이 작업은 Adapter statistic을 알아내는 방법이다. IOCTL_PROTOCOL_QUERY_OID .와 방법은 유사하다.

IOCTL_PROTOCOL_RESET

이 작업은 Adapter를 초기화 해주는 방법으로 거의 쓰이지 않는다

IOCTL_PROTOCOL_READ
이 작업은 네트웍에서 Packet을 가져와 읽어오는 방법이다.

```
int RcvPacket(HANDLE hVxD,
              BYTE* Buffer,
              DWORD cbIn)
{
    HANDLE      hEvent;
    DWORD       cbRet = 0;
    OVERLAPPED  ovlp    = {0,0,0,0,0};
    int         result;

    hEvent = CreateEvent(0, TRUE, 0, NULL);

    if (!hEvent)
        return SYSERR;

    ovlp.hEvent = hEvent;

    result = DeviceIoControl(hVxD,
                             IOCTL_PROTOCOL_READ,
                             Buffer,
                             cbIn,
                             Buffer,
                             cbIn,
                             &cbRet,
                             &ovlp);

    if (!result)
        GetOverlappedResult(hVxD, &ovlp, &cbRet, TRUE)

    CloseHandle(hEvent);

    return cbRet;
}
```

Note: 인자 cbIn 는 이더넷이므로 1514 이다.

IOCTL_PROTOCOL_WRITE

이 작업은 Packet을 네트워크를 통해서 보내는 방법으로 IOCTL_PROTOCOL_READ 와 유사하다.

Note: 패킷은 이더넷 헤더 형식을 완전하게 갖추어야한다.

IOCTL_PROTOCOL_MACNAME

이 작업은 네트워크 어댑터의 이름을 문자열로 저장하는 방법이다.

```
result = ControlPacket((HANDLE) etptr->handle,
                       IOCTL_PROTOCOL_MACNAME,
                       iBuf,
                       32,
                       iBuf,
                       32);
if (result > 0) {
    memcpy(arg1, iBuf, result);        /* arg1 의 형식은 char * */
    return OK;
}
return SYSERR;
```

IOCTL_PROTOCOL_BIND

이 작업은 특정 어댑터에 VPACKET 을 바인딩하는 방법이다.

ControlPacket

함수의 내부구현은 다음과 같다.

```
int ControlPacket(HANDLE hVxD,
                  ULONG ioctl,
                  BYTE* inBuffer,
                  DWORD cbIn,
                  BYTE* outBuffer,
                  DWORD cbOut)
{
    HANDLE      hEvent;
    DWORD       cbRet = 0;
    OVERLAPPED  ovlp    = {0,0,0,0,0};
    int         result;

    hEvent = CreateEvent(0, TRUE, 0, NULL);

    if (!hEvent)
        return SYSERR;
```

```
    ovlp.hEvent = hEvent;

    result = DeviceIoControl(hVxD,
                             ioctl,
                             inBuffer,
                             cbIn,
                             outBuffer,
                             cbOut,
                             &cbRet,
                             &ovlp);

    if (!result)
        GetOverlappedResult(hVxD, &ovlp, &cbRet, TRUE);

    CloseHandle(hEvent);

    return cbRet;
}
```

나. RCVCAPTURE

RVCCAPTURE는 VPACKET으로부터 들어오는 패킷을 처리하는 코드이다. 네트워크 카드의 심볼을 인자로 넘겨주면 그것으로 들어오는 모든 패킷을 수신할 수 있게 한다. Network card의 specification은 registry의 HKEY_LOCAL_MACHINE₩System ₩CurrentControlSet ₩Services ₩Class ₩Net에서 찾을 수 있다

WindowsNT에서는 먼저 Packet32.sys를 네트워크 프로토콜에 등록시켜 주어야한다. 네트워크 프로토콜에 Packet32.sys를 등록시키기 위해서는 제어판에서 네트워크를 선택한 후 Protocol 탭에서 추가를 눌러 RcvCAPTURE 디렉토리에 있는 Oemsetup.inf를 지정하여주면 된다.

(1). 구 조

(가). Vpacket의 생성

먼저 CVpacket의 pointer를 선언한 후 Windows의 platform에 맞추어 CVpacket95 또는 CVpacketNT를 동적으로 할당을 해준다.

```
switch (GetOsVersion())
{
case VER_PLATFORM_WIN32_WINDOWS:
        g_pVpacket = new CVpacket95;
        break;

case VER_PLATFORM_WIN32_NT:
        g_pVpacket = new CVpacketNT;
        break;
```

```
default:
        printf("not support platformWn");
        exit(1);
}
```

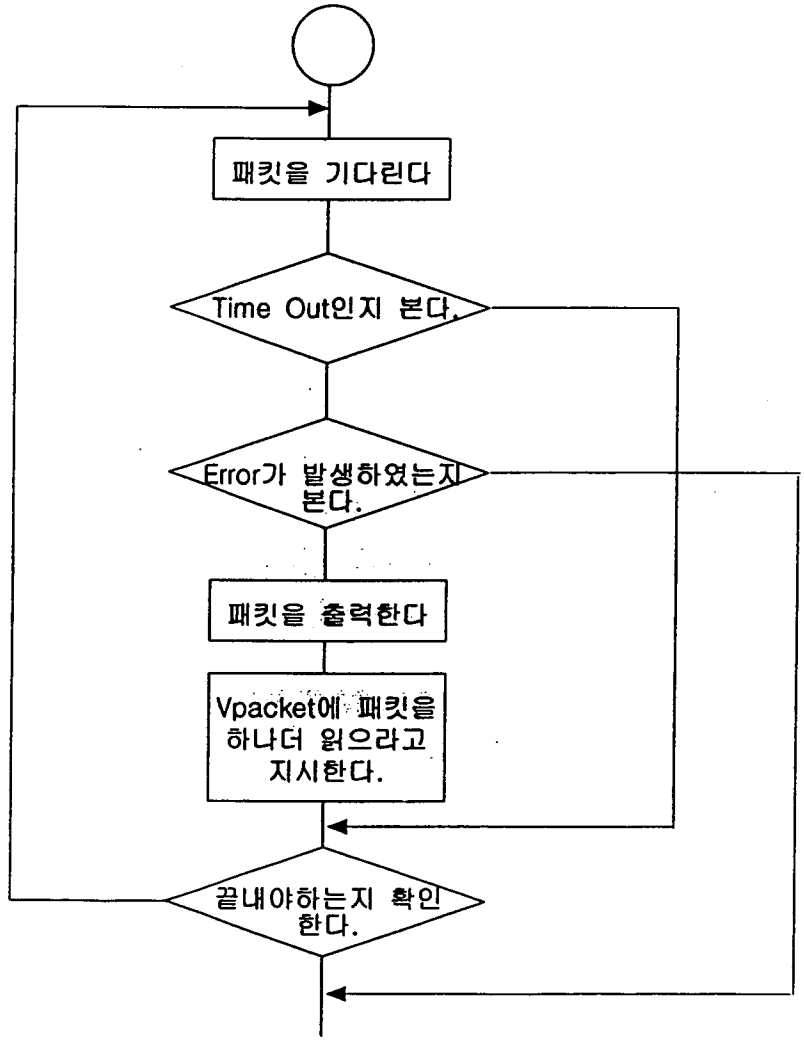
(나). Vpacket의 초기화

Vpacket이 Protocol Driver를 load하고(LoadVpacket), Windows95일 때는 특정한 Network card를 지정하여 Binding 시킨다(BindNIC).

그리고, Network card로 부터 모든 패킷을 받을 것을 지정하고 (SetAsPromisMode), 패킷을 받을 버퍼를 초기화(InitVariable)하면 Vpacket이 패킷을 읽을 준비는 끝난다.

(다). 패킷 읽기

패킷을 읽는 것은 다음과 같은 흐름으로 이루어진다.



[그림 4-14] 순서도

이를 code로 표현하면 아래와 같다.

```
do
{
        // packet이 들어올 때까지 기다린다.
        do
```

```
        {
                nIndex = g_pVpacket->WaitForPacket();
        }
        while (nIndex == WAIT_TIMEOUT && g_bContinue);

        if ((nIndex & 0xffffffc0) == 0)
        // packet이 들어온 것이다.
        {
                // packet을 가져온다.
                pop = g_pVpacket->GetPacket(nIndex);

                ////////////////////////
                // packet을 출력한다.
                ...
                ////////////////////////

                // packet을 다시 driver에 단다.
                if (!g_pVpacket->ClingBufferToVxD(nIndex, FALSE))
                {
                        printf("fail to cling buffer to driver₩n");
                        g_bContinue = FALSE;
                }
        }
        else if (nIndex != WAIT_TIMEOUT)
        // packet을 받는데 error가 발생하였다.
        {
                printf("wait failed!₩n");
                g_bContinue = FALSE;
        }
}
while (g_bContinue);
```

(라). Unload

Ctrl-c handler를 달아서 사용자가 패킷을 받는 것을 멈출 수 있게 한다. 사용자가 Ctrl-c를 누르면 g_bContinue가 false의 값을 가지고 위의 패킷 읽는 작업을 그만둔다. 패킷 받는 것을 멈추면 Vpacket을 Unload한다.

6. JAVA로 구현한 Packet Capturing Program 구축

가. 프로그램 개요

본 프로그램은 자바를 이용하여 LAN을 통한 패킷을 수집하여 모니터링을 하는 프로그램이다. 자바가 갖는 프로그래밍 언어상의 장점으로 본 프로그램은 시스템에 의존적이지 않으며 다양한 환경의 시스템에서도 사용할 수 있는 잇점이 있다.

프로그램은 크게 다음의 세 가지 구성으로 나누어질 수 있다.

(1) 암호 메시지 사용자 감시 기능.

근거리 통신망 내 모든 TCP/IP 패킷을 실시간으로 수집, 분석하여 암호화된 메시지가 전송되는 송, 수신 IP주소를 출력한다.

(2) 특정 IP 주소 감시 기능.

감시자가 지정한 IP주소에 대하여 LAN을 통한 모든 송, 수신패킷을 감시하여 실시간으로 송수신되는 메시지를 수집, 분석한다. 그리하여 수집된 메시지를 분석하여 메시지의 패턴을 리포팅한다. 만약 현재 송수신되는 패킷 데이터가 평문인지 암호문인지를 판단하며 평문일 경우는 이전에 저장된 데이터와 비교하여 현재 송수신되는 패킷이 어떤 유형의 파일 타입인지를 나타내 주며, 또한 암호문일 경우에도 연구 결과에 의하여 어떤 유형의 파일 타입이 암호화 되었는지를 추정하여 보고한다.

(3) 각종 패턴 데이터의 관리기능

일종의 자가 학습기능으로 프로그램이 수행되면서 얻어지는 결과값을 기존에 구축된 데이터에 추가로 등록하여 이전 데이터와의 비교 분석으로 새로운 패턴 데이터를 구축한다. 또한 관리자가 지정하는 파일을 선택하여 지정된 파일 타입을 신규로 등록을 하여 관리할 수 있다.

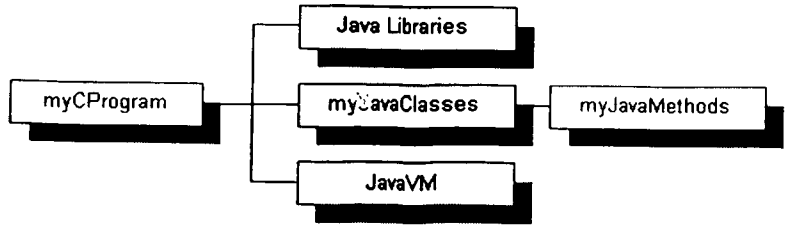
나. JNI

프로그램이나 라이브러리를 작성하는데 자바가 모든것을 해결하지는 못한다. 하부시스템의 깊숙한 곳까지 다루는 프로그램을 작성하려 할 때, 하부시스템을 보다 잘 다루는 언어로 작성할 것이다. 이러한 상황을 돕기 위해 자바는 native메소드인 JNI를 제공한다.

JNI 는 Java Native Interface 로 Java 와 Native Code와 연계할 수 있는 프로그래밍 인터페이스로 제작된 JDK 의 일부 API 이다. 기존의 C, C++, 그리고 어셈블리 등으로 작성된 코드를 자바와 연계하여 사용할 수 있게 한다.

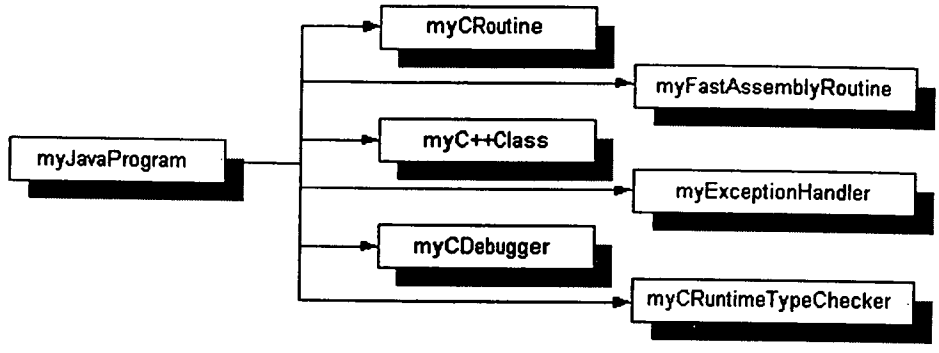
JNI는 운영체제에서 지원하는 동적 라이브러리 개념을 사용하고 있다. 이는 프로그램에 직접 링크되지 않고 필요시 메모리에 적재되어 실행되는데 윈도우즈와 솔라리스와 같은 운영체제 등에서 다양하게 지원한다.

[그림 4-15]은 레가시 C 프로그램을 JNI 라이브러리 이용하여 자바의 메소드, 클래스 등을 호출하는 것을 간략히 나타낸 것이다.



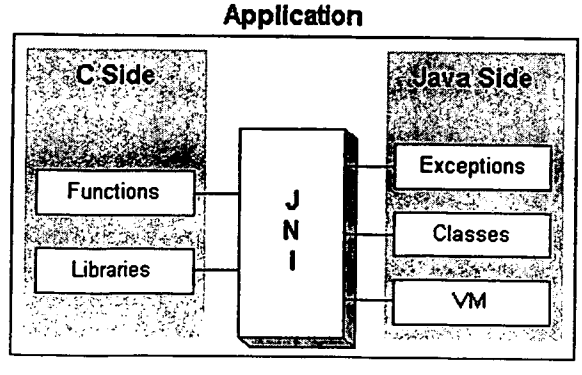
[그림 4-15] JNI 호출

[그림 4-16]는 자바에플리케이션에서 native language 함수나 클래스를 를 호출하는 것이다. 이것이 가능한것은 JNI 구현으로 이루어진다.



[그림 4-16] native 호출

[그림 4-17]처럼 JNI는 C로 구현된 영역과 자바로 구현된 영역을 손쉽게 연동하여 준다.

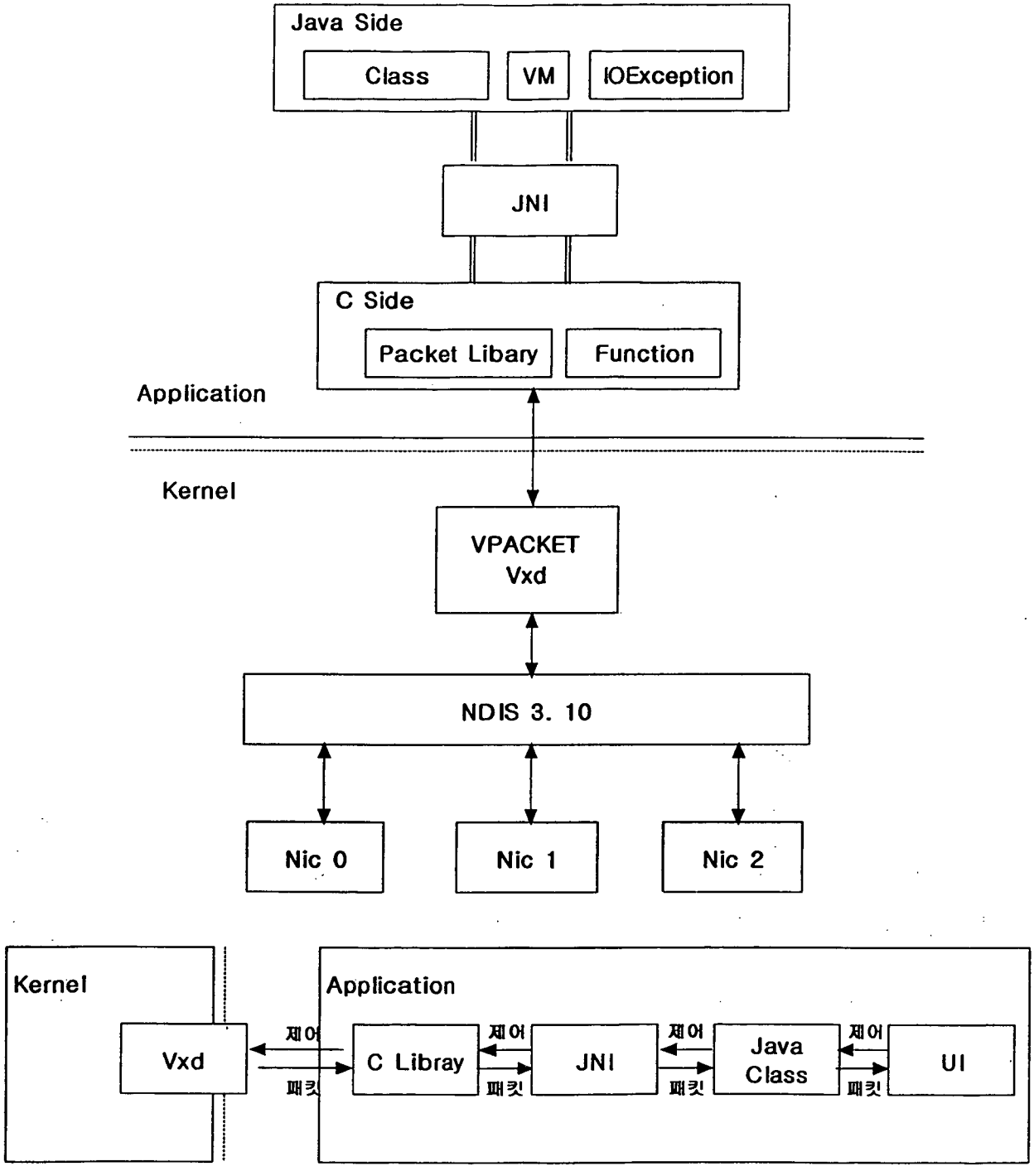


[그림 4-17] JNI 연동

위에서 살펴본 바와 같이 본 연구에서는 자바로 네트워크를 감시하는 프로그램을 구현하는 과정에서 하부시스템 깊숙히 관여하는 부분에서는 C++로 구현된 라이브러리를 사용하였다. 유닉스계열의 시스템에는 수 많은 네트워크관련 라이브러리가 있어 폭넓게 사용할 수 있지만 WIN32 프로그래밍 환경에서는 하위 레벨의 네트웍 접근을 할 수 있는 직접적인 방법을 제공하지는 않는다. 따라서 하위 레벨의 네트웍 접근을 해야 하는 어플리케이션은 일반적으로 VxD라고 부르는 Virtual Device Driver 을 사용하였다.

다. 시스템 구성도

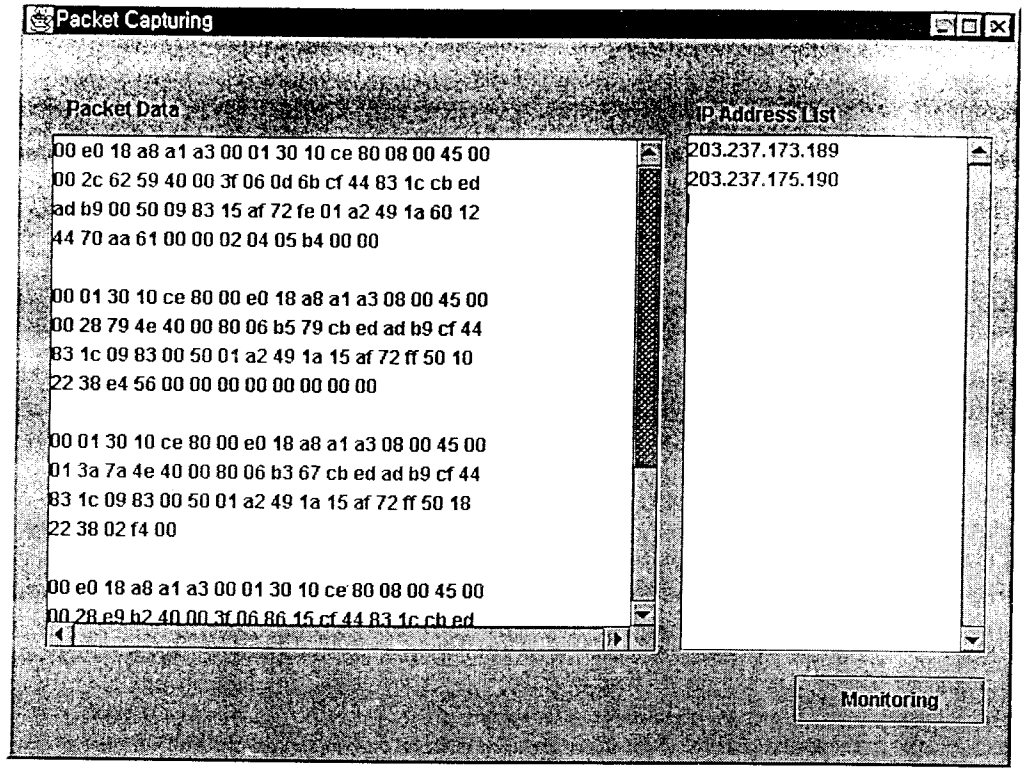
다음은 위에서 언급한 Vxd와 JNI와 본 프로그램의 시스템 구성도이다. 다음 구성도는 Win32시스템에서 구현된 구성도이다.



[그림 4-18] 시스템 구성도

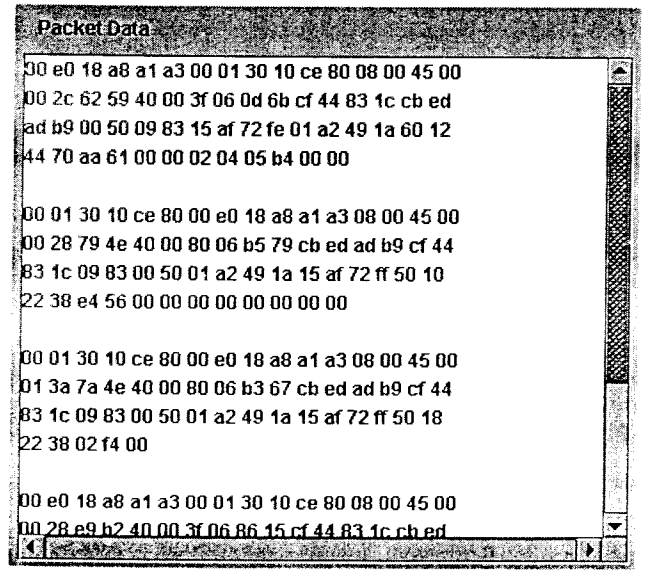
7. 프로그램 사용설명

가. 암호메시지 사용자 감시 기능



[그림 4-19] 실시간 모니터링 프로그램 화면

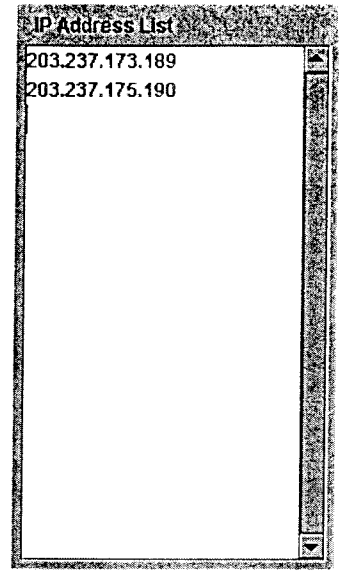
모니터링 버튼을 클릭하면 LAN을 통한 모든 패킷을 수집한다. 수집한 데이터는 Packet Data의 텍스트필드에 [그림 4 - 20]와 같이 나타난다.



[그림 4-20] 수집된 패킷

Packet Data에 적재되는 텍스트들은 패킷 단위당 나타내는 16진수 값으로 표현된다. 시스템에서 정한 패킷크기를 기준으로 수집되어지는 모든 패킷데이터를 분석한다. 데이터량0

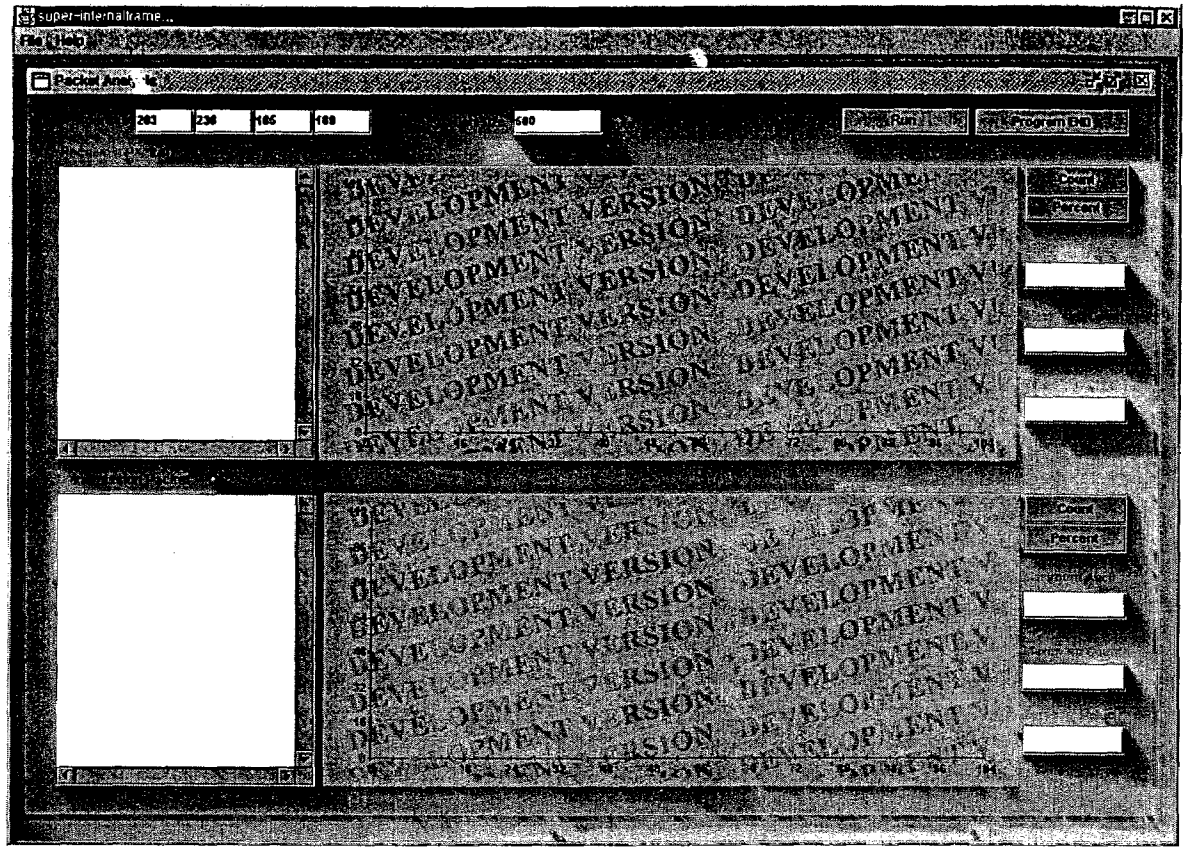
적은 패킷은 분석에서 누락된다. 즉, 분석할 수 있는 최소량의 데이터를 가지고 있는 패킷을 얻는다.



[그림 4-21] IP 주소

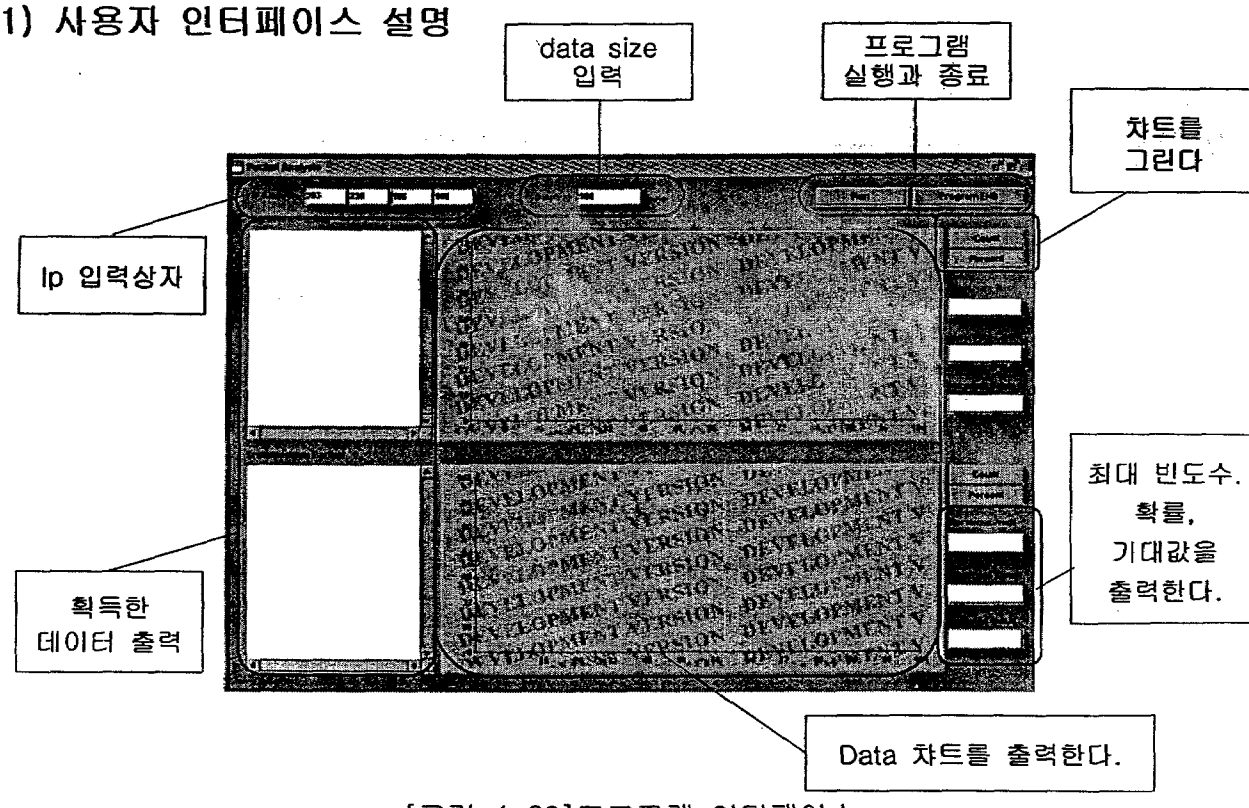
[그림 4-21]에 적힌 IP주소는 시스템이 판단한 암호 메시지를 송, 수신하는 IP의 주소이다. 시스템 감시자는 위에 적힌 IP주소를 가지고 다음장에 나올 특정 IP주소 감시 기능을 이용하여 보다 세밀한 감시를 가능하게 할 것이다.

나. 특정 IP address 감시 기능



[그림 4-22]프로그램 초기화면

(1) 사용자 인터페이스 설명

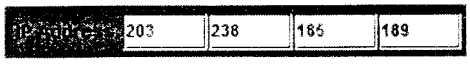


[그림 4-23]프로그램 인터페이스

(2) 패킷 획득 및 분석

(가) 획득할 패킷의 조건 입력

① Ip 입력



대상 호스트의 Ip를 텍스트 박스에 입력한다.

입력된 Ip address 는 취득한 헤더의 Ip와 검사한 후 일치하면 패킷을 저장한다.

② 데이터 크기 입력

입력된 크기만큼 데이터를 받은 후, 더 이상 패킷을 취득하지 않으며, 사용자가 지정한 크기 만큼의 데이터를 가지고 차트를 그린 후 기대값을 산정한다.

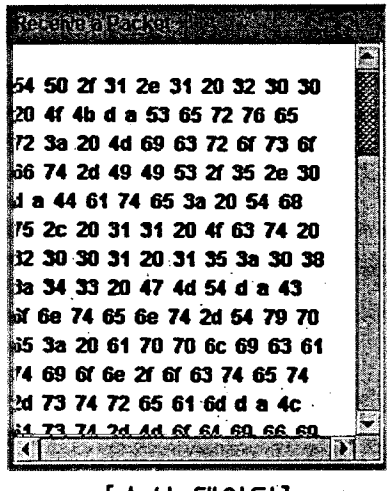
조건을 입력한후 Run 버튼 클릭

(나) 패킷 획득

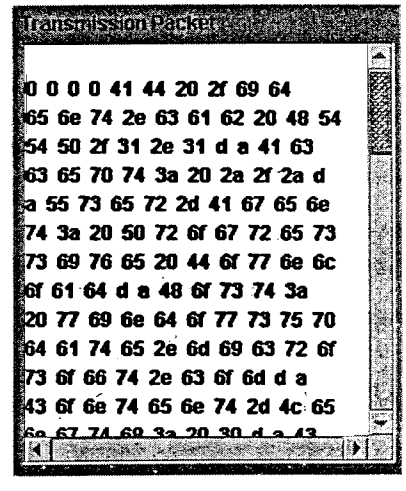
획득된 데이터는 [그림 4-24]와 같이 16진수 데이터로 받아들여진다.

Receive Packet : 지정한 호스트에서 수신하는 데이터를 출력한다.

Transmission Packet : 지정한 호스트에서 송신하는 데이터를 출력한다.



[수신 데이터]



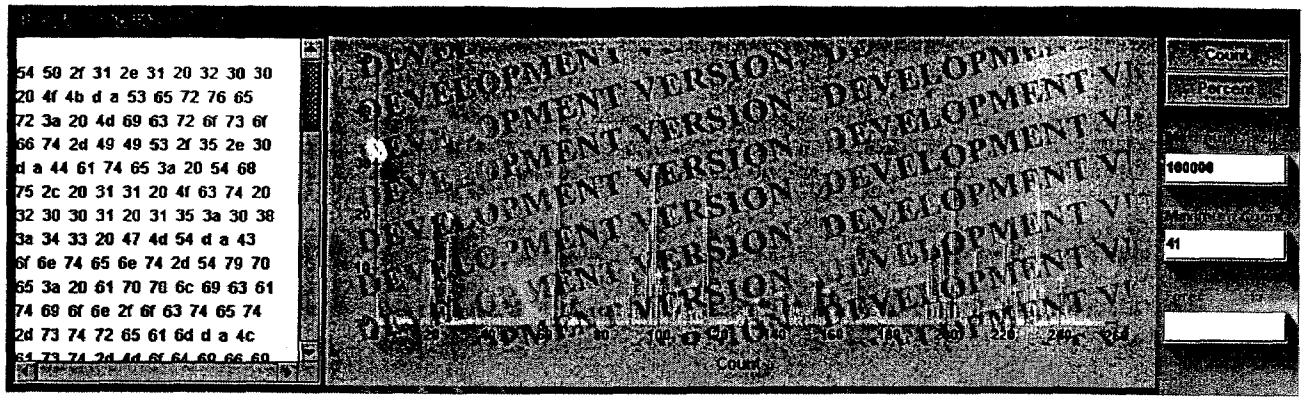
[전송 데이타]

[그림 4-24]획득된 데이터

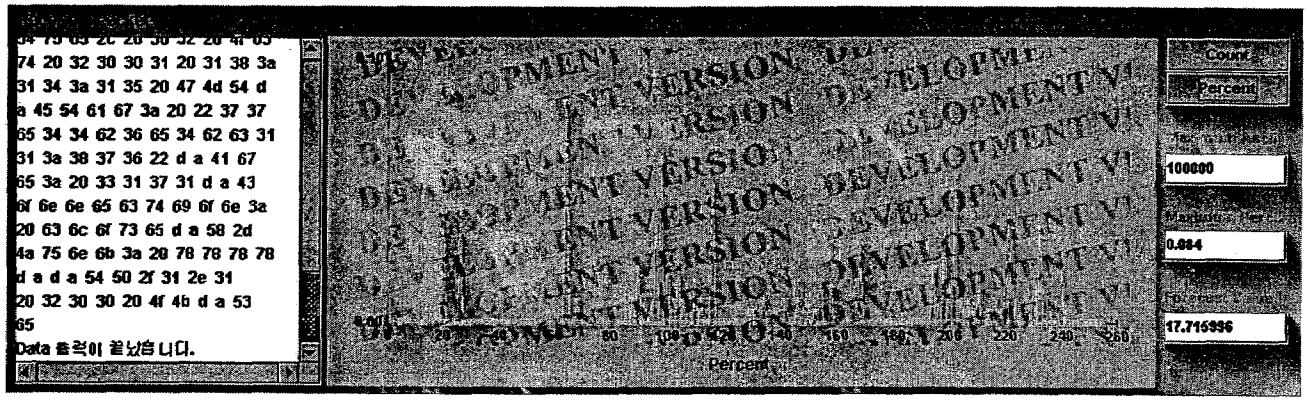
(다) 차트 그리기

획득된 데이터로 차트를 그리려면 Count, Percent 버튼을 클릭하면 각각의 차트를 출력한다.

해당 호스트에서 취득된 데이터는 [그림 4-25]와 같이 X축은 아스키 값을 나타내고 Y축 값은 빈도수와 확률을 나타낸다.

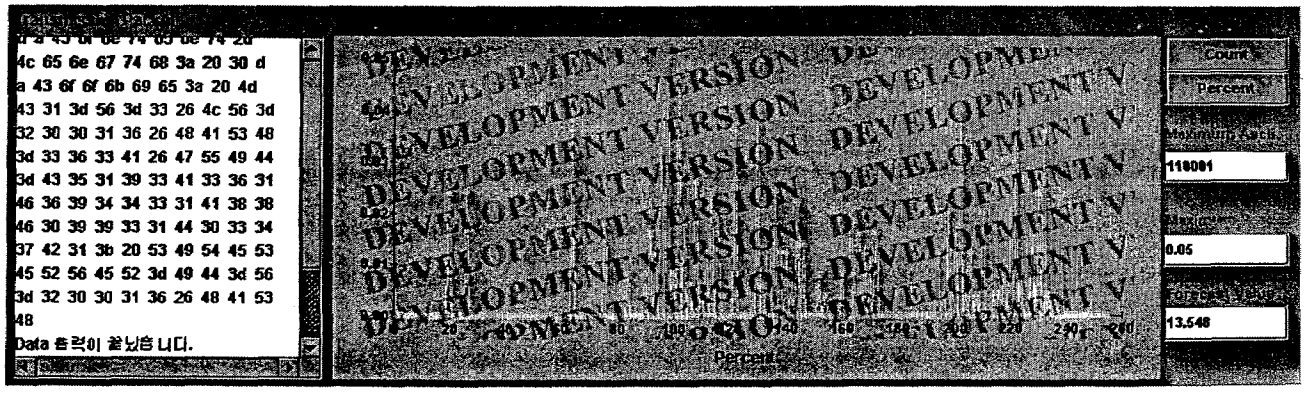


[수신 데이터 카운트]



[수신 데이터 확률]

[전송 데이터 카운트]

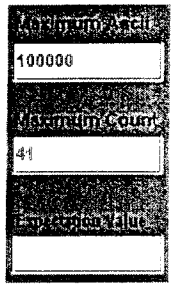


[전송 데이터 확률]

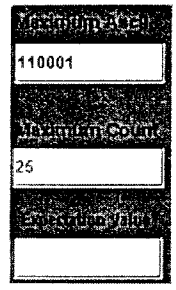
[그림 4-25]차트

(라) 최대 아스키, 최대 빈도, 최대 확률, 기대값 출력

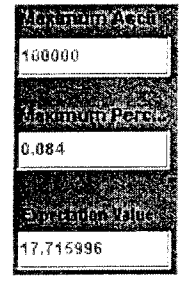
Maximum ASCII Code : 가장 많은 빈도수, 가장 높은 확률을 보인 아스키값을 출력한다.
Maximum Count : 가장 높은 빈도수를 출력한다.
Maximum Percent : 가장 높은 확률을 출력한다.
Expectation Value : 받아들인 패킷의 파일 종류를 판별하는데 사용되는 기대값이다.



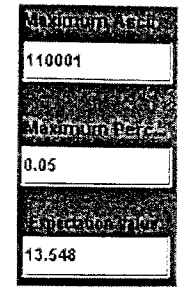
[수신카운트]



[전송카운트]



[수신확률]



[전송확률]

[그림 4-26] 아스키, 빈도수, 확률, 기대값

(3) 데이터 분석을 통한 파일 종류 판별

위와같이 얻어진 데이터를 통하여 암호문인지 평문인지를 판별하는 것은 이미 3장에서 설명하였다. 본 프로그램은 3장에서 언급된 알고리즘과 로직을 자바로 구현한 것이다.

제 5장 결론

1. 연구의 결론

본 연구는 인터넷의 보편화와 무선 통신 기술의 보급에 따라 군사적인 목적으로 개발되어 사용되던 암호화 기술이 상업용과 일반소비자용으로 사용되기 때문에 생겨날 수 있는 사회적 역기능을 해결하기 위해 수행되었다. 즉, 암호화 기술을 이용하여 범죄집단이 메시지를 주고받는 경우 국가 기관이 이를 해독할 수 없으며 국가 주요 기밀문서가 외부로 유출되는 경우에도 이를 막을 장치가 없다는 문제점을 해결하기 위해 수행되었다.

마약, 테러, 유괴 등의 범죄에 가담한 집단의 통신데이터나 저장 데이터를 해독하기 위해서는 (1) 우선 평문과 암호문의 구분을 정확히 할 수 있어야 하며 (2) 암호문의 경우 사용된 암호화 알고리즘을 판별하고 (3) 가능하다면 작성된 문서의 종류를 구분할 수 있어야 한다는 것이 본 연구의 기술적인 목표로 수립되었다.

평문과 암호문의 경우, 고전적인 문자 분포 통계 데이터를 사용하여서도 가능하지만 주파수 공간에 투영하였을 경우의 correlation 측정으로도 가능해진다. 본 연구에서 시도된 correlation 측정법은 메시지의 길이가 300Byte이상인 경우 100%의 정확도를 보였으며 300Byte이하인 경우에도 100Byte 이상이면 97%이상의 정확도를 보였다.

본 연구의 가장 중요한 목표는 암호문이 경우 패턴 분석으로부터 사용된 암호화 알고리즘을 판정(Identify)하는 것이다. 본 연구의 사전 연구에서 대칭 키 알고리즘인 DES, IDEA를 비교하였을 경우 recurrence Plot이나 return map을 통해서 확연하게 다른 패턴이 나타났기 때문이다. 이는 대단히 중요한 결과로서 현재까지 대부분의 암호 분석 연구들이 암호 키를 찾아내기 위한 무차별 공격(Brute-Force Attack)에 매달리고 있다는 점을 생각하면 대단히 획기적인 연구결과가 얻어질 수 있는 가능성이 있었다. 그러나 되풀이되는 실험을 통해 사전 연구의 결과가 잘못되었다는 점을 확인하였다. 이는 프로그램의 오류로서 결과적으로 암호화 알고리즘들간의 패턴 구분은 암호문에서는 불가능하다는 것이 본 연구의 결론이다.

이처럼 암호화 알고리즘간의 차이가 생성된 암호문을 통해서 관찰되지 않는 것은 기본적으로 모든 암호문이 Random Number를 생성하도록 구조가 만들어져 있기 때문이다. 첫번째 시도로서 약 52,000개의 일반 데이터를 암호화하고 이를 8Byte 블럭 단위로 복호화 하여 생성된 시계열 데이터를 return map이나 recurrence plot

으로 하였으나 알고리즘별 차이는 관찰되지 않았다. 이를 바꾸어 생성된 블록에서 1Byte 단위로 복호화하여 return map과 recurrence plot을 그려보았으나 역시 패턴은 나타나지 않았다. Hurst Index 측정 등의 여러 가지 시도를 통해서도 같은 결과가 얻어졌으면 결론적으로 대칭형 키에서는 암호화된 문장으로부터 사용된 알고리즘을 구분하는 것이 불가능하다는 결론에 도달하게 되었다.

다음으로는 암호문이라고 판정되는 경우, 그 암호문이 원래 어떤 파일로 만들어졌는지를 구분하기 위한 연구가 진행되었다. 이외로 암호화 알고리즘에 대해 원시 파일의 파일 형식에 따라 각기 다른 패턴이 생성됨을 알 수 있었다. 즉, 암호화된 문장으로부터 그 원래 파일이 MS Word, Power Point, 아래 한글, 엑셀, Zip압축, Jpeg압축 등의 구분이 가능함을 알게 되었다.

2. Contribution

본 연구를 통해서 암호화된 문장의 알고리즘 판정은 불가능하다는 결론을 얻게 되었으며 암호화된 문장의 원래 파일 형태에 대한 판정은 통계적인 방법으로 가능하다는 것을 알게되었다. 이는 비록 본 연구에서 목표로 하였던 암호화 알고리즘 판정은 불가능하지만 원 파일에 대한 약간의 정보를 추출할 수 있음을 의미한다.

본 연구에서는 네트워크를 패킷 분석을 통해 실시간으로 감시하고, 만약 암호화된 메시지라고 의심되는 경우 이를 분석하여 원래 메시지의 파일 형태를 구분해 내는 기술은 개발되었다. 따라서 국가기관에서 테러, 유괴, 마약, 매춘, 탈세 등의 범죄집단이 사용하는 통신문이나 통화문을 분석하여 암호문이라고 의심되는 메시지에 대해서는 이를 분석하거나 집중적인 감시를 할 수 있는 기술이 개발되었다. 물론 실시간으로 암호문을 해독하는 것은 어느 기술을 하더라도 현재로서는 불가능하다. 그러나 적어도 작성된 암호문이 원래 어떤 파일 형태였는지는 파악할 수 있게 되었다.

3. 기술의 활용

본 연구에서 개발하려는 기술은 미국의 FBI, CIA, NSA 등의 법 집행기관들이 암호기술의 일반화로 대단히 갈구하는 기술이다. 물론 연구 제안 당시 목표로 하였던 암호 알고리즘의 구분은 현재의 기술, 즉 주파수 분석이나 Fractal 분석, recurrence map, return plot, 등을 사용하여서는 불가능하다는 것이 밝혀졌다. 그러나 본 연구에서 얻어진 암호화된 문장의 원래 파일 형태 파악은 어느 정도의 유용성을 갖는다고 생각된다. 예를 들어 일반문서와 달리 엑셀 문서는 도표용으로, 도면이나 사진과 같은 영상 데이터는 주요 설계도 혹은 사진용으로 사용되기 때문에

문서의 크기와 파일의 종류가 파악되면 범죄집단의 감시와 동태 파악이 쉬워질 수 있을 것으로 생각한다.

본 기술은 국가 정보원, 국군 기무사, 경찰 등에서 유용하게 사용될 수 있을 것으로 생각된다.

[참고문헌]

1.Biham, E., A. Biryukov, N. Ferguson, L. Knudsen, B. Schneier, and A. Shamir, Cryptanalysis of Magenta, Second AES Candidate Conference, April 1999.

2.Coppersmith, D., D. Wagner, B. Schneier, and J. Kelsey, Cryptanalysis of TwoPrime, Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, 1998, 32-48.

3.Ferguson, N., and B. Schneier, Cryptanalysis of Akelarre, Fourth Annual Workshop on Selected Areas in Cryptography, August 1997, pp. 201-212.

4.Ferguson, N., B. Schneier, and D. Wagner, Security Weaknesses in Maurer-Like Randomized Stream Ciphers, Fifth Australasian Conference on Information Security and Privacy (ACISP 2000).

5.Ferguson, N., J. Kelsey, B. Schneier, M. Stay, D. Wagner, and D. WHiting, Improved Cryptanalysis of Rijndael, Seventh Fast Software Encryption Workshop, Springer-Verlag, 2000.

6.Hall, C., J. Kelsey, V. Rijmen, B. Schneier, and D. Wagner, Cryptanalysis of SPEED, Fifth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1998, pp. 318-338.

7.Kocher, P., Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.

8.Kocher, Paul, Joshua Jaffe, and Benjamin Jun, " Differential Power Analysis," in advances in Cryptology-Crypto 99, Springer LNCS, v1666, pp388-397;a brief version was presented at the rump session of Crypto98.

9.Kohno, D., J. Kelsey, and B. Schneier, Preliminary Cryptanalysis of Reduced-Round Serpent, Third AES Candidate Conference, 2000.

10.Kelsey, J., B. Schneier, and D. Wagner, Key-Schedule Cryptanalysis of 3-WAY, IDEA, G-DES, RC4, SAFER, and Triple-DES, Advances in Cryptology--CRYPTO '96 Proceedings, Springer-Verlag, August 1996, pp. 237-251.

11.Kelsey, J., B. Schneier, and D. Wagner, Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA, ICICS '97 Proceedings, Springer-Verlag, November 1997, pp. 233-246.

12.Kelsey, J., B. Schneier, D. Wagner, and C. Hall, Side Channel Cryptanalysis of Product Ciphers, ESORICS '98 Proceedings, Springer-Verlag, September 1998, 97-110.

13.Kelsey, J., B. Schneier, and D. Wagner, Key Schedule Weakness in SAFER+, Second AES Candiate Conference, April 1999.

14.Kelsey, J., B. Schneier, and D. Wagner, Mod n Cryptanalysis, with Applications Against RC5P and M6, Fast Software Encryption, Sixth International Workshop Proceedings (March 1999), Springer-Verlag, 1999, pp. 139-155.

15.Kelsey, J,. and B. Schneier, MARS Attacks! Preliminary Cryptanalysis of Reduced-Round MARS Variants, Third AES Candidate Conference, 2000.

16.Kelsey, J., T. Kohno, and B. Schneier, Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent, Seventh Fast Software Encryption Workshop, Springer-Verlag, 2000.

17.Wagner, D., L. Simpson, E. Dawson, J. Kelsey, W. Millan and B. Schneier, Cryptanalysis of ORYX, Fifth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1998, pp. 296-305.

18.Wagner, D., N. Ferguson, and B. Schneier, Cryptanalysis of FROG, Second AES Candidate Conference, April 1999.

19.Wagner, D., B. Schneier, and J. Kelsey, Cryptanalysis of the Cellular Message Encryption Algorithm, Advances in Cryptology--CRYPTO '97 Proceedings, Springer-Verlag, August 1997, pp. 526-537.

20. http://www.kisa.or.kr