Secure Internet-based Electronic Commerce: The View from Outside the US

Peter Gutmann

Department of Computer Science
University of Auckland
pgut001@cs.auckland.ac.nz

Introduction

Because of a number of well-publicised computer break-ins there has been a steadily increasing demand for encryption and related security measures to be included in software products. Unfortunately these measures often consist either of "voodoo security" techniques where security is treated as a marketing checkbox only, or are rendered ineffective by the US governments refusal to allow non-Americans access to the same security measures which it allows its own citizens, making the current electronic commerce infrastructure a target-rich environment for attackers. Organisations employing such (in)security systems may make themselves liable for damages or losses incurred when they are compromised. This paper covers the issues of using weak, US government-approved security as well as problems with flawed security measures, examines some of the measures necessary to provide an adequate level of security, and then suggests several possible solutions.

This paper is targeted at people with a responsibility for computer security as well as those currently considering the extent to which their organisation may wish to become involved in Internet commerce, and includes fairly extensive coverage of past and present Internet commerce related security problems in order to given a general idea of areas to look out for. Although little security knowledge is assumed, some sections are intended for more technically-aware readers and may be skipped if desired.

Problems in Internet-Based Electronic Commerce

The creation of a global electronic commerce system will provide an extremely powerful magnet for hackers, criminals, disgruntled employees, and hostile (but also "friendly") governments' intelligence agencies. This problem is magnified by the nature of the Internet, which allows attackers to quickly disseminate technical details on performing attacks and software to exploit vulnerabilities. A single skilled attacker willing to share their knowledge can enable hordes of dilettantes around the world to exploit a security hole in an operating system or application software within a matter of hours [Gordon 1994]. One example of how easy these tools make it for neophytes to attack a system involved someone gaining super-user privileges on a Unix system and then trying to execute DOS commands. The Internet also enables an attacker to perform attacks over long distances with little chance of detection and even less chance of apprehension. The ability to carry this out more or less anonymously, at low cost, and with little chance of being caught, encourages attackers.

In order to see where an attack may take place, it is necessary to examine the nature of traffic on the Internet. Sending a message over the net is much like writing it on the back of a postcard. Once the card has been filled, the networking software adds a sender and destination address and sends it over the network, with the rest of the message being continued on other postcards. Each transmitted card is combined with cards from other systems and moved in the general direction of the destination system. At each point in the network, a router sorts the cards and forwards them in the appropriate direction. The cards can be damaged in transit (in which case they are resent), and may travel over multiple different routes depending on traffic conditions in various sections of the network. Since a route isn't known in advance, it's impossible to control which hosts the cards pass through[1], so the sender of the cards has to hope that all the hosts in the path are operated by trustworthy people who won't try to modify the cards, or add forged cards, or disclose the contents of the cards to others.

[1] It is possible to use the IP protocols source routing facility to explicitly specify the path for each data packet, but since this opens up various security holes it is generally disabled.

The specific features which are most useful to protect commercial transactions on the Internet are identification (proof that both sides are who they claim they are), confidentiality (the inability of outsiders to read the message traffic), non-repudiation (the ability to prove that someone really did send a particular message), and, in the case of financial transactions, protection against replay attacks (the inability to repeatedly send the same message such as a purchase order). Various other features such as proof of message submission, proof of message delivery, message flow confidentiality, and so on, can also be useful. In addition, auditing and accounting features are essential, as will be shown later.

The Internet was originally created as a research tool and not as a commercial environment. As a result it was designed for ease of use and collaboration rather than security, with provisions allowing for sharing of files and information on a trust basis. Simple security measures were available, but were usually a side-effect of the Unix environment in which most of the software operated. The lack of security on the Internet ranged from the lowest levels (no protocol in the entire TCP/IP suite contains any authentication or confidentiality features, making it possible to impersonate users or hosts, and intercept and modify data) through to the top-level protocols (FTP has only simple password-based authentication and is often configured to allow anonymous, non-password-protected connections, and the ease with which the SMTP mail protocol can be exploited is legendary). Although the next version of the Internet protocols addresses these flaws [Atkinson 1995a] [Atkinson 1995b] [Atkinson 1995c] [Metzger 1995] [Karn 1995] (as well as many other problems which have plagued the Internet for some time, see [Hinden 1996] for an overview), the deployment of these new protocols will probably take quite some time, and many implementations may only be available in a crippled form due to US government export restrictions. Other approaches to securing the Internet are given in [Bhimani 1996].

Traditional vs. Internet-based Electronic Commerce

In the last few years the growth of the Internet because of the popularity of the world-wide web has made the prospect of online electronic commerce very attractive for businesses. Because the Internet is fast becoming all-pervasive, it allows even smaller businesses easy access to previously inaccessible markets and provides consumers with the ability to do business with companies which were previously unavailable to them. Unfortunately the cooperative research environment of the Internet is ill-suited to this mode of operation. In order to allow secure commercial use, applications using the Internet have to bring their own security mechanisms along with them. This problem was mostly avoided by the OSI telecommunications model, which was heavily influenced by monopoly PTT's (phone companies) who were capable of imposing security "solutions" by fiat. In the OSI world all communications were to be carried out over links run by and through switches operated by the PTT's, which were defined to be secure, eliminating any need for separate encryption and authentication mechanisms. Unfortunately this model was not well suited to a pluralistic telecommunications environment, and the Internet gradually took over and relegated the value-added network (VAN) providers to a niche market. The reasons for this were many and varied, but typically included cost (sending a byte of data over an X.25 link in New Zealand costs approximately 400 times as much as sending it over the Internet), easy accessibility (opening an account with an Internet service provider (ISP) typically requires a modem, a phone line, and a valid credit card), and easily available, often free software (most operating systems now come with built-in TCP/IP support and associated application software). By eliminating the need to invest in expensive, proprietary EDI translation or mapping software and specialised communications hardware, the Internet grants even smaller firms easy access to global markets.

Traditional EDI security relies on bilateral trading agreements to establish mutually-accepted policies which cover various threats, rather than actually securing the transaction. The threat of receiving false or altered orders is supposedly mitigated by complex rules and regulations which detect and invalidate anomalous transactions. Although some EDI security measures exist, very few organisations seem to be using them. A general solution to the problem of securing EDI has been to wrap it up in an external security layer such as PGP or S/MIME (Secure MIME), which makes it both secure and "Internet-compatible" in one operation. At the other end of the connection the data is unwrapped and forwarded to the EDI software which processes it as usual, unaware that a strong security layer has been added to the exchange.

A recent study on the use of email-based financial EDI carried out on behalf of the Bank of America and Lawrence Livermore National Laboratory shows that this is a viable and very cost-effective alternative to traditional EDI mechanisms, with little message transmission overhead and (rather surprisingly) no lost messages (although there were occasional delays due to hosts going down, network outages, and similar problems). In addition, as the trial progressed the overall problem count decreased

as various issues were ironed out. Unfortunately the number of messages transmitted was very low (although a single message might contain 1000 or more payment instructions), so that some of the figures may be a bit misleading. For larger volumes of messages it would be preferable to use online transaction systems rather than store-and-forward, email-based ones. The main reason for basing the study on an email-based system was that at the time there was no software available to perform online Internet-based EDI [Segev 1996].

As a result, an electronic commerce system will typically consist of a large number of clients communicating with a merchant over the Internet, and then a single link from the merchant to a clearing house over a more traditional medium such as an X.25 link. One report estimates that in 1997 56% of all banking transactions will be made through ATM's and EFTPOS networks, home banking, and telephone service centers, with the main growth areas being telephone and Internet-based banking [Head 1995]. In Europe roughly 80% of the Internet access market consists of high-speed business links, with the 12.5 million business users who use the Internet to escape the tariffs which burden other communications media far outweighing 2.5 million home users [Fox 1996]. This paper's particular focus on Internet-based electronic commerce security problems doesn't mean that non-Internet electronic commerce is any better, it is simply subject to less scrutiny which means that the gaping security holes which often exist are never brought to anyone's attention and therefore never fixed (for example one NZ government department transfers very sensitive records over X.25 links with an encryption system which uses the time of day, with a one-second granularity, to choose a per-message encryption key. Although this provides only 86,400 possible keys, they make it easier for an attacker by including the time in the unencrypted header of the message). As will be shown below, public scrutiny and analysis server to make a security system stronger, not weaker.

Data Interception

The best target for an attacker intent on gathering as much of other peoples' data as possible is either at the routers used to forward data packets over the net, or at the ISP used to connect to the Internet. Someone intent on carrying out a surreptitious attack can take advantage of the fact that most local area networks work as "spy networks" in which each node watches the traffic flowing over the network and only picks out messages intended for it [McLeod 1993]. It takes only a minor modification such as placing an ethernet card into promiscuous mode to allow one machine on a network to intercept information for all machines on that network. Since a low-cost 486 PC is capable of filtering all traffic on a typical T1 (1.544Mbps) link [BorderWare 1995], a single PC can provide the capability of monitoring all data sent over all but the fastest links.

The way many ISP's are run can affect their overall security. Governments and the military rely on techniques such as security clearances, special background investigations (SBI's), and compartmentalisation for security. Businesses rely on employee contracts and non-disclosure agreements. ISP's on the other hand are often run on a very informal basis, and are staffed by both paid employees and "random people" who turn up to help in exchange for free network access. System administrators tend to be chronically overworked, underpaid, and underappreciated by management (there is a fairly active Usenet newsgroup, `alt.sysadmin.recovery`, devoted largely to this topic). Since system administrators can spend weeks at a time working in "fire-fighting" mode, they often have little time available to check that every single security precaution has been taken, and that no supplemental functionality has appeared in the network drivers of a machine connected to the network. This, combined with the fact that ISP's concentrate a large number of users at a single point, make them very tempting targets for data interception. This has been confirmed by the fact that a number of large backbone ISP's have been the subject of sniffing attacks. Incidents of packet sniffer use are reported almost daily to computer security monitoring groups [CERT 1995].

Accidental leakage of data can also be a problem. With the inevitable collection of misconfigured routers, bridges, protocol tunnels, and other loopholes which appear in a network over time, combined with the use of sophisticated routing protocols which go to great lengths to get data through using any method available to them, data packets can turn up in very unusual places. The author is aware of one case in which payroll data from a large organisation leaked over an internal X.25 network, across to an IP network with nonstandard and unusual network addresses and routing configurations, over a bridge to a dial-up WAN link, back into an X.25 network, and finally over a router into another IP network where it appeared in the debugging logs on a firewall. This was possible because various gateways, bridges, and protocol tunnels, along with a few misconfigured setups, had combined to propagate the data in ways which came as a considerable surprise to the organisation from which it was originating.

Other organisations have also found strange and unusual data packets appearing on their networks [Bellovin 1993].

This kind of leakage may be introduced deliberately by an attacker. For example a large network organisation in New Zealand until recently used the same password on all its routers for both low and high-level security access for ease of maintenance, allowing anyone who could guess it (it was a very obvious password) to reroute traffic and intercept data from other ISP's links. Since this is very difficult to detect without either manually reading through what are often multimegabyte routing tables or analysing data packets as they travel over the network, and once in place will continue operating without any further maintenance, installing this form of covert network wiretap is a highly desirable goal for an attacker. Even protection mechanisms such as firewalls can often be bypassed by employing "stealth scanning" techniques in which certain features of the TCP/IP protocol such as the manipulation of protocol status flags can be exploited to bypass firewalls which block certain packets. These techniques may even crash some routers which aren't equipped to handle the unusual packet types.

Attacks on routers don't necessarily require access to the router itself. For example an attacker could use Internet Control Message Protocol (ICMP) redirect messages to subvert routing and ensure messages travelled over a path which made them easy to intercept, or do the same thing indirectly by using selective ICMP unreachable messages to force messages to be sent over alternate paths or to alternate hosts. A typical active attack (in which the attacker does more than just monitor traffic moving over a network) would involve setting up a mirror of an existing web site and transparently redirecting connections to the mirror rather than the original. Without using a security protocol such as SLL, there is no way for the end user to tell that they are connecting to a false clone of the original site.

Bulk interception of traffic is also possible, but difficult. A 1977 report commissioned by the US government contained a catalogue of surveillance and interception techniques for intercepting microwave and coaxial cable transmissions on a large scale [Mitre 1977], but because of the cost involved this sort of interception is really only feasible for governments [Hager 1996] although it may be carried out on a small scale by anyone with access to the communications link such as phone company employees [Hyde 1976]. The main effect of the report was, like the adverse publicity about Netscape's security, to spur companies to build better security into their systems. For example IBM began adding encryption facilities to its private telecommunications systems after the report was made public with the result that many IBM facilities both in and outside the US now encrypt data and phone traffic [Horgan 1985].

A large list of Internet-based attacks covering exploits, impersonation, data-driven and infrastructure attacks, and social engineering, is given in [Ranum]. Other techniques for competitive information acquisition are given in [Kahaner 1996].

Classification of Threat Types

Discussions of attacks on security systems have typically focused on a single style of attack motivated either by intelligence-gathering requirements or criminal profit. In reality there are two further classes of attack above and beyond the basic criminal attack which pose a far greater threat to a security system designed to protect electronic commerce. The three classes of attack are:

1. The criminal attack. The motivation for this attack is "How can I acquire the maximum financial return by attacking the system?". The attacker will typically try to use the minimum necessary resources for the maximum possible gain. These attacks are generally opportunistic and will focus on low-tech flaws in a system rather than sophisticated analysis and attacks.

2. The publicity attack. The motivation for this attack is "How can I get the most publicity by attacking the system?". The attacker will typically have access to significant resources (sometimes millions of dollars worth of equipment) via research institutions or corporate networks and large amounts of time, but few financial resources.

3. The legal attack. The motivation for this attack is "How can I discredit the system to prove my clients innocence?". The attacker will have all the resources of a publicity attack, but may also have significant financial support.

The focus to date has been almost entirely on criminal attacks, although the media have often taken publicity attacks and reported them as though they were being applied as criminal attacks (see, for

example, the SSL encryption breaking saga on page 7). However it is questionable whether some publicity attacks are feasible when applied as criminal attacks. For example consider the oft-cited liabilities involved in protecting credits cards sent over the Internet with weak encryption. Recovering a single credit card number requires intercepting a message on its way from a client machine to a server and then investing perhaps a week of idle computing time on a small collection of computers to break the encryption if the card number is protected with weak exportable encryption. All this effort results in the recovery of a single credit card number. A far simpler method is to "shoulder surf" at a sales counter, or to visit the garbage bins behind a store (particularly one where valuable purchases are made, such as a jeweller) and recover the discarded credit card slips, which contain the card number, name, expiry date, and customers signature. Instead of recovering information on one card in a week, information on dozens of cards can be recovered in minutes and, if the store is chosen carefully, the cards will have a high probability of having fairly sizeable credit limits. Other ways of acquiring credit card information involve "social engineering" techniques which are often as simple as directly asking for the details [2600 1993] (an example of a trick which has become popular recently is to place an enticing white-collar job offer in a large newspaper and then apply for credit cards using the personal details provided by applicants). Fairly extensive coverage of problems related to credit and ATM card fraud is given in [Clough 1993]. Card fraud along these and similar lines accounted for US$500M in losses in 1991 in Europe alone [Banker 1993], with VISA losing US$655M worldwide (0.2% of turnover) in 1994 [Norton 1994], although this appears to be slowly dropping as new security techniques are introduced [Cards International 1994]. Since the intent of a criminal attack is to acquire the maximum financial return for the minimum investment in effort, it seems unlikely that intercepting Internet traffic and breaking the encryption on it will be seen as a profitable enterprise by the criminal element when compared to other types of bank and credit-card related fraud [BAB 1992], although there are exceptions to this rule — one organisation is seriously looking at the prospect of selling van Gogh's over the Internet, a transaction which would be a tempting target for a criminal attack.. In fact a search of online and media resources failed to locate a single documented case of credit card information being compromised by intercepting a transmission and/or breaking the encryption on it, although there were a number of cases of attackers breaking into improperly secured computer systems and acquiring sizeable databases of user accounts and credit card information, in one case involving details of 20,000 users and cards [NYT 1995a] (this is not so say that transmitting credit cards over the Internet is secure, merely that noone has yet reported any problems).

Of far greater concern to organisations considering Internet-based electronic commerce are publicity and legal attacks. Publicity attacks involve considerable expertise on the part of the attacker, but are often undercapitalised. Legal attacks don't suffer from this restriction. These types of attacks are closer to the military threat model in which an opponent is determined to destroy you no matter what the cost. In a criminal attack motivated solely by profit, this isn't feasible; in a legal attack it is. The standard computer security threat model developed by the military assumes sophisticated analysis and penetration attempts against well-run, well-defended systems, but this model tends to fall down when applied to an attack by a student with access to a million-dollar supercomputer against a product put together in a series of late-night hacking sessions and weekend overtime work in order to meet a release schedule, which is how a typical publicity attack might be mounted.

Political Impediments to Strong Security Systems

Cryptography, for centuries used mainly for diplomatic purposes and by the military, has in the last few years moved into the commercial world. Strong encryption software is available from virtually any country in the world, can be typed in from books available in bookstores, is taught in university mathematics and computer science courses, and includes algorithms so simple they can be implemented in about 10 minutes by anyone with the necessary typing skills[2]. Unfortunately encryption policy is still ruled largely by the military and intelligence agencies, who see it as a threat to their intelligence-gathering mission. By defining encryption technology to be munitions, the US government brings it under the umbrella of arms control laws [DOC 1980] [DOS 1989] [Root 1991] [DOS 1992] and restricts the export of encryption technology to implementations which have been weakened to the extent that US intelligence agencies (along with foreign governments, companies, criminals, and students with too much time on their hands) have no trouble in breaking it. Citizens usually rely on the government to look after their safety interests in areas such as transport and medicine where they lack

[2] The algorithms used were RC4 and TEA, taken from a book available in a local bookstore. The test subject was 12 years old.

the knowledge to make proper evaluations, but when it comes to computer and communications security the US government is doing just the opposite.

In terms of enabling the large-scale gathering of intelligence through communications interception, this strategy has proven very effective. A recent joint Department of Commerce/NSA study found that the overwhelming majority (about 75%) of general-purpose software products such as word processors, spreadsheets, and database software available outside the US is of US origin, that the US has few viable foreign competitors for such products, and that of the general-purpose products which contain encryption, all were of US origin (the effectiveness of the encryption in these products is discussed on page 26) [DOC 1995]. Since the software can't be exported if it contains strong encryption, the world has to make do with weak or no security. Unfortunately the lack of security aids not only the US governments surveillance requirements, but also any foreign governments, criminals, and hackers with an interest in other people's data.

Effects of US Export Restrictions

Because creating two separate versions of a program with US and non-US levels of security is difficult and expensive, many US software producers have either not sold their products outside the US, or created a single product with weak encryption for sale inside and outside the country. The discouraging of strong encryption has had the effect of hindering the development of security solutions provided by US firms and, because the US does not have a monopoly on good cryptography, opening the market for non-US firms who are free to sell strong encryption technology worldwide (including inside the US) in a market protected for them by the US government [Levy 1996]. The US market for encryption software was US$384M in 1991 rising to an estimated US$946M in 1996, with the world market rising from US$695M in 1991 to an estimated US$1.8B in 1996 [Hoffman 1994]. Another report has estimated that by the year 2000, US firms will be losing US$30-60B in sales to overseas competitors if the current policy remains unchanged [CSPP 1996], and this figure in itself pales into insignificance compared to the amount which will be lost through industrial espionage and other criminal activity which is made possible because the required cryptographic protection isn't available. In addition the inability to sell strong encryption overseas has a collateral damage factor when other software deals fall through because a crucial encryption component can't be provided. US companies have reported losing numbers of overseas accounts to foreign competitors because they couldn't deliver software capable of withstanding even a modest key recovery attack: "The laws are punishing US companies, and we're losing business to foreign countries because they can offer the same thing. The law is not holding back the flow of encryption, it is just holding back US companies from making money" [Dunlap 1996].

Because of the massive loss in revenue due to the restrictions on strong encryption, the US government has come under increasing pressure to lift the embargo and allow the export of strong encryption software. The governments' response was a string of "key escrow" systems which required producers of encryption hardware and software to hand over the encryption keys to the US government before they could sell their products. Unfortunately encryption technology which was conducive to US government spying [Madsen 1994] [Madsen 1995] [Reuter 1995] was seen as undesirable by non-US users, and unmarketable by US companies; the resulting debate has been documented in great detail elsewhere [Hoffman 1995] [Gutmann 1996c]. Ironically, even when a security system was specifically designed to provide a backdoor for the US government, its export has not been allowed. Although backdoor access (and therefore exportability) was the primary design goal of the NSA's own Clipper/Capstone/Fortezza encryption hardware, not only can the hardware not be exported from the US but even the programming details required to write software to control it can't be released to non-US citizens. Similarly, Mastercard and VISA were stopped from distributing the reference implementation of their SET financial transaction technology outside the US despite the fact that the entire system had been designed specifically to meet the US governments export requirements and could only be used for protecting financial transactions [Lewis 1996].

Recent Developments

A recent US government report recommended the lifting of export controls because of the damage they were causing to US business and commerce in general [NRC 1996], and several bills have been introduced to Congress to try to amend the encryption export laws. So far this has resulted in only one minor change in the status quo: The US export restrictions were recently amended to allow limited export of encryption software for personal use [Federal Register 1996] after people who tried to follow the previous regulations discovered that even the people charged with enforcing them didn't understand

them or know how to deal with them [Blaze 1995] [Plumb 1995], and in tacit recognition of the fact that this form of export couldn't really be controlled anyway.

However in other areas efforts to suppress strong encryption have been as strong as ever. Within the last year the NSA has threatened companies and programmers with criminal charges for merely providing software which is capable of interoperating with encryption software, but which itself doesn't contain a single line of encryption code (sometimes referred to as "code with a DES-shaped hole"). This has included the suppression of a PGP interface for an email application [Wirbel 1996] and a PEM interface for a web server [NCSA 1995]. Even devices such as TV set-top boxes have been blocked from export on the grounds that they are munitions [Robertson 1996] [NYT 1996]. The response to this extremely restrictive environment has been for some US companies to move encryption development overseas, for example Sun to Russia[3] [Battelle 1995] and RSADSI to China and Japan [Corcoran 1996].

In November 1996, in a game of crypto musical chairs, the control over encryption software was moved by an executive order — a presidential decree — from the Department of State to the Department of Commerce [Clinton 1996][4]. The changeover was used to rewrite significant portions of the regulations to make them not easier to work with as was generally expected, but instead to make them even more restrictive. The regulations were extended to cover entirely new areas of technology such as anti-virus software and firewalls. One possible explanation for this is that by restricting the entire range of products a computer security company is likely to produce (even if these products have nothing to do with encryption), the US government can gain further leverage with these companies when persuading them to build back doors for the US government into their software, a requirement which is covered in great detail in the new regulations. An alternative explanation is that by restricting any good security products for US use only, the government is helping to weaken the information infrastructure of other countries while at the same time allowing US companies to hold the strong hand in a potential "information war".

The Netscape SSL Break and its Implications

The Secure Socket Layer (SSL) protocol, after a somewhat shaky start (version 1 was broken within 10 minutes of being unveiled [Hallam-Baker 1996]), and an attempt by Microsoft to promote a similar but competing protocol [Benaloh 1995], has more or less edged out any other protocols to become the standard for securing HTTP sessions (an overview of SSL and various other proposed WWW security mechanisms is given in [Reif 1995a]). SSL uses a combination of RSA and, usually, a proprietary (until it was reverse-engineered, of which more later) algorithm called RC4 to provide confidentiality, data integrity, and authentication. Since it was built into what was by far the most popular web browser, and because of Netscape's policy of giving away the software, it immediately gained widespread popularity, providing the means for establishing a pervasive low level security mechanism across all IP protocols. No details on RC4 were published, but the fact that it was designed by a very good cryptographer was enough to reassure most people. RC4 is used in dozens of commercial products including Lotus Notes, a number of Microsoft products such as Windows for Workgroups, Windows 95, Windows NT, and Access, Apple's AOCE, and Oracle Secure SQL.

The main criticism of SSL (apart from a few protocol flaws which were fixed in later versions) was the fact that RC4 used a key of only 40 bits, making it susceptible to a brute-force key recovery attack. The reason for the 40-bit key and (according to RSADSI, the company that developed RC4) the reason why details on it were kept secret was that these conditions were required under an agreement between the Software Publishers Association (SPA) and the US government which gave special export status to the RC4 algorithm and a companion algorithm called RC2 [RSA Labs 1995]. Implementations of RC2 and RC4 which are restricted to a 40-bit key get automatic export approval provided the implementations work correctly with a set of test vectors supplied by the NSA. If the results are as expected, export approval is granted within a week, a process which is vastly easier than the arcane maneuvers necessary to obtain export permits for other algorithms (if they are granted at all).

---

[3] The very country the export restrictions are ostensibly supposed to stop encryption technology from getting to.

[4] These executive orders are issued due to "an unusual and extraordinary threat with respect to which a national emergency has been declared". The actual national emergency which is used to regulate encryption in this manner changes from time to time, and ranges from rogue nations and terrorists through to the lack of a law which has the same effect as the executive order. The original national emergency which was used appears to be the Korean War.

The weakness of the encryption in US-exportable SSL implementations even led the French government, which normally bans all non-government-approved use of encryption[5] to approve the use of Netscape in France [Vincent-Carrefour 1996], presumably because the French government has no problems in breaking it. Other software was deemed to be exportable if the encryption was compromised in an even more severe manner, for example version 3 of Lotus Notes had to have both the conventional encryption weakened to 40-bit keys *and* the public-key encryption weakened to 400-bit keys, although only one of the two was necessary (anyone who can factor a 400-bit public key can recover all the 40-bit session keys, and anyone who can recover 40-bit session keys doesn't need to factor the 400-bit public key).

Reverse-Engineering RC4

The first step in attacking SSL was to find out how RC4 worked. Since it was in widespread use, it was only a matter of time before someone picked the code apart and published the algorithm. RSADSI sell a cryptographic toolkit called BSAFE [BSAFE 1994] which contains RC4, and this seems a likely source for the code (the Windows password encryption code is also a good source, and the algorithm can be extracted in an hour or two). The results were posted to mailing lists and the Internet [Anon 1994b]. Someone with a copy of BSAFE tested it against the real thing and verified that the two algorithms produced identical results [Rescorla 1994], and someone else checked with people who had seen the original RC4 code to make sure that it had been (legally) reverse-engineered rather than (illegally) copied [Anon 1994c]. RC4 turned out to be an extremely simple, but also very elegant cipher:

```
while( length-- )
    {
    x++;
    sx = state[ x ];
    y += sx;
    sy = state[ y ];
    state[ y ] = sx;
    state[ x ] = sy;
    *data++ ^= state[ ( sx+sy ) & 0xFF ];
    }
```

The RC2 code was disclosed in a similar manner in 1996, but after problems with legal threats during the RC4 disclosure process it was handled more formally: First an RC2 implementation was reverse-engineered [Anon 1996a], then a specification for the algorithm was written based on the reverse-engineered code [Gutmann 1996a], and finally a new implementation based on the specification was written by someone who had never seen the reverse-engineered RC2 code, creating a proper clean-room copy [Vogelheim 1996]. No legal threats were ever issued over RC2.

The RC4 code was immediately subject to intense analysis in various cryptography-related fora. RC4 has two parts, the initialisation phase, and the random number generation phase used for the encryption itself. Initially, an array is initialised to be a random permutation using the user's key, and then the random number generator mixes the permutation and reports values looked up pseudorandomly in that permutation.

Among various RC4 problems which were discussed are that during the initialisation phase small values may remain in small positions in the initial permutation; user keys are repeated to fill 256 bytes, so `aaaa` and `aaaaa` produce the same permutation; results are looked up at pseudorandom positions in the array, and if some internal state causes a certain sequence of positions to be looked up, there are 255 similar internal states that will look up values in the same sequence of positions (although the values in those positions will be different), from which it can be shown that cycles come in groups of 2^n, where all cycles in a group have the same length, and all cycles are of an odd length $\times$ 256 unless they are in a group of 256; there is a bias in the results so that, for example, the pattern "`a a`" is too likely and the pattern "`a b a`" is too unlikely, which can be detected only after examining about 8 trillion bytes; the

[5] The "decret du 18 avril 1939" defines 8 categories of arms and munitions from the most dangerous (1st category) to the least dangerous (8th category), the "decret 73-364 du 12 mars 1973" specifies that encryption belongs to the second category, and the "loi 90-1170 du 29 decembre 1990" states that use of encryption equipment must be approved by the French government. The head of the French DGSE has stated that they have a policy of supplying French industry with information obtained through their interception of international telecommunications which have helped them to obtain major international contracts [Marion 1991].

internal state is not independent of the results, so that with a given result there are two patterns in the internal state that appear $^1/_{256}$ times more often than they ought to; at least two separate methods exist for deducing the internal state from the results in around 2^{900} steps; and for $^1/_{256}$ keys there is a 13.8% chance that the initial byte of the pseudo-random stream generated by RC4 is strongly correlated with only a few bytes of the key.

All of these "weaknesses" except for the last one are purely theoretical in nature, and even the last one can only occur under special circumstances (it doesn't affect SSL implementations since they hash the key with MD5 rather than using it directly, which avoids the problem). Overall, the cryptographic community agreed that RC4, when used correctly, was a sound cipher.

Unfortunately, due to the US export restrictions, RC4 couldn't be used correctly. Although Netscape negotiated a 128-bit key to protect each session, it sent 88 of those 128 bits in the clear so that only 40 bits of the key were actually kept secret. Now that RC4 was known, SSL became a prime target for a key recovery attack. An initial attempt at breaking RC4 was made in early July 1995 using encrypted data from a Microsoft Access database [Back 1995a]. This attempt involved 89 contributors and took about a week using idle computing time on workstations and PC's, with around 80% of the work being done by the top 19 contributors. Due to logistical problems, human error, and buggy software, the attempt ultimately failed, but the stage had been set for an attack on SSL.

The Attack on RC4/40

On 14 July 1995, an SSL challenge message containing an encrypted (fake) credit card order transmitted to one of Netscapes own computers was posted to mailing lists and the Internet by Hal Finney [Finney 1995a]. The challenge message was independently broken by two groups, the first to announce success in breaking it was a French researcher using idle time on a collection of 120 computers and workstations over 8 days [Doligez 1995a] [Doligez 1995b] [Trei 1995]. The 40-bit secret part of the key was `7E F0 96 1F A6`, and was found after scanning just over half the key space. The average search speed was about 850,000 keys/s, with a peak of 1,350,000 keys/s. A second group had broken it two hours earlier, but announced their success a day later [Back 1995b]. The event immediately attracted international media attention, including newspaper, radio, and television coverage (although many reports were rather garbled) in France [Munger 1995], Germany [Reif 1995b], Japan [NewsBytes 1995], the UK [Arthur 1995], and the US [Beck 1995] [Sandberg 1995a].

A second challenge was posted on 19 August 1995 [Finney 1995b] and a key recovery attack by a 'Brute Squad' of 201 Internet-connected volunteers began at 1800 GMT on 24 August 1995. The attack involved greatly improved software which progressed through the key space in an order chosen to maximise the possible amount of MD5 precomputation which could be done, and included automatic communication between client workstations attacking the encryption and a central server running the recently designed Simple Key Search Protocol which doled out sections of keyspace to search [Brooks 1995]. This setup took 31.8 hours to find the key, `96 36 34 0D 46`. Congestion on the server being used to coordinate the attack meant that most of the machines involved were idle for perhaps 75% of their available time, so in theory the attack could have been completed in only 8 hours. Both the client and server software were continually upgraded during the duration of the attack (the server went through 7 software upgrades while it was running).

The key recovery attacks, which used only unused processing capacity on the machines, were essentially "free", and could easily be mounted using the spare processing capacity available to companies, businesses, universities, and foreign governments. By breaking a brute-force attack of this kind into a number of independent sections, as many machines as are needed can be applied to the problem, so that each doubling of the amount of hardware applied to the problem halves the time required to find the solution. Although the total investment will have doubled, the cost per recovered key is kept constant since twice as many keys can now be found in the same time.

Another possibility which has been suggested is the creation of a key recovery screen saver for networked Windows machines which performs key searches during the (often prolonged) periods in which machines are left idle. One experiment in performing this kind of attack took 2 weeks using relatively slow 486/50's and Sparc 20's, with noone the wiser that the machines were being used overnight for this purpose [Young 1996b]. Another attack involved a networked Windows screen saver where the client software was activated whenever a machine was otherwise idle and communicated its results to a central server on a network with around 100 PC's. By now, breaking the Netscape

encryption had become a kind of processor benchmark, with one manufacturer rating the speed of their system based on how long it took to break RC4 — 8 hours on one computer [ICE 1996].

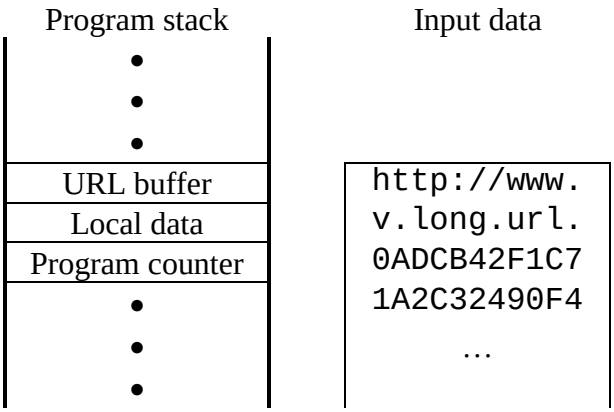
The latest version of the SSL protocol specification now recommends that implementations requiring strong security refuse to operate with 40-bit keys [SSL 1996]. SSL 3.0 also addresses most of the complaints about problems in earlier versions of the protocol, leaving only a small number of relatively minor and easily-fixed problems [Wagner 1996].

Further improvements to the attack were proposed. The most important one was to move from attacking one message at a time to attacking entire collections of messages. Instead of generating a key and testing it against a single message, it could be tested against 100 messages, so that in average one key could be found in $^{1}/_{100}{}^{th}$ the time it took for a single message. Unfortunately in the case of SSL this wasn't possible, since although only 40 bits of key are kept secret, there are still a total of 128 unique key bits for each message, making it impossible to attack more than one message at a time. In effect the remaining 88 bits of key act as a 'salt' in the same way the Unix password salt works. However a more simplistic implementation which uses only 40 bits of key could be attacked in this manner. More details on attacks on short session keys are given on page 18.

The attacks on RC4 are a prime example of a publicity attack. They were carried out by volunteers using borrowed machine time, noone was harmed (strangely enough, even Netscapes stock prices were almost unaffected), and they achieved a great deal of publicity. The intended goal — of proving that the restricted encryption allowed by the US government could be broken — was achieved. This fuelled intense debate within the US about the need to lift the export restrictions in order to facilitate electronic commerce. Virtually every article covering the encryption debate would eventually refer to the ease with which the 40-bit keys of the form used in SSL could be broken and the negative impact this had on business confidence (see for example [Ante 1996] [Markoff 1996a] [Markoff 1996c] [NYT 1995b] [USA Today 1995]). The fact that it was uneconomical to mount a criminal attack on 40-bit SSL keys was mostly ignored (except in Netscape press releases). The enthusiasm for Internet commerce, especially commerce protected by SSL, was severely dented, and a number of companies began to reconsider their rush to deploy commercial services over the net. Other organisations such as the Bank of Montreal reacted by disallowing any transactions with 40-bit keys and giving away web browsers with 128 bit encryption, an option which was unfortunately only open to US organisations (or at least organisations operating in the US) doing business with US clients [Majer 1996].

(In)Secure Internet Electronic Commerce

After the RC4 attacks, researchers looked for other weaknesses in Netscape. In September 1995 it was discovered that Netscape didn't check the amount of input it was fed, leading to internal buffers being overrun [Green 1995] [Neumann 1995] [Sandberg 1995c]. This bug also existed in other browsers such as Mosaic and IBM's WebExplorer. By carefully adjusting the data fed to the browser, it was possible to force a victims PC to execute arbitrary code simply by selecting a URL. This problem has occurred in the past in a number of other programs such as `fingerd` (where it was exploited by the Internet worm [Spafford 1988]), the CERN and NCSA `httpd`'s (as explained on page 26), and in the time this paper was being prepared in `splitvt`, `syslog`, `mount`/`umount`, `sendmail`, `lpr`, `bind`, `gethostbyname()`, `modstat`, `cron`, `login`, `sendmail` again, the `query` CGI script, `newgrp`, AutoSofts RTS inventory control system (a whole slew of programs), `host`, `talkd`, `getopt()`, `sendmail` yet again, FreeBSD's `crt0.c`, WebSite 1.1, `rlogin`, `term`, `ffbconfig`, `libX11`, `passwd`/`yppasswd`/`nispasswd`, `imapd`, `ipop3d`, SuperProbe, `lpd`, `xterm`, `eject`, and `lpd` again, leading one exasperated mailing-list moderator to wonder how many more times he'd see this problem [Bloodmask 1996]. The flaw in question can be exploited by ensuring that the code to be executed is located in the URL at a point where it overflows the end of the buffer. A URL can contain almost any data value except for a few characters which have special significance and a binary zero, a restriction which can easily be bypassed by selecting alternative encodings for any instructions which cause problems. The browser stack (with an excessively long URL containing data which affects the programs execution) looks as follows:



By making the URL long enough to overwrite the other data and saved program counter it is possible to force a jump to an attackers code rather than returning to the calling routine, allowing an attacker to force the execution of arbitrary code on the victims machine. Although this exploit is machine and browser-specific, by targeting the most common architecture (Windows on an Intel processor) and browser (Netscape), a reasonable chance of success can be obtained.

Random Number Generation

At about the same time the stack overwrite problem was discovered, a basic flaw was found in Netscape's SSL implementation which reduced the time to break the encryption from hours to minutes. Despite an existing body of literature covering the need for carefully selected random-number generation routines for cryptographic applications [Eastlake 1994] [IEEE 1995] [Robertson 1995], one of which even included ready-to-use code [Plumb 1994], Netscape used fairly simple methods which resulted in easy-to-guess encryption keys [Goldberg 1995] [Goldberg 1996a]. It was found that, under Unix, Netscape used a combination of the current time in seconds and microseconds and the process ID of the current and parent process to initialise the random number generator which produced master encryption keys. The time can be determined to a reasonable degree of accuracy from the message being sent, the process ID's can be determined using standard Unix utilities (by people using the same machine that Netscape is running on), or by using other tricks such as the fact the an approximate process ID can often be obtained by observing the output of other network-related programs on the machine, and the parent process ID is often 1 (for example when Netscape is started from an X-windows menu) or close to the process ID. Under Windows the implementation was similarly flawed.

The resulting number of values to search are smaller than the number of combinations in a 40-bit key, and much smaller than the number of combinations in the 128-bit key in the export-restricted version. Since Netscape never reseeds its internal random number generator, subsequent connections are relatively easy to break once the first one is broken. The researchers who discovered this problem released a program, `unssl` [unssl 1995] which would break Netscape's encryption (both the weak exportable version and the strong export-restricted version) in about a minute on an average workstation. Although one of the researchers classed it as "a silly bug", it received large amounts of media attention, including front-page coverage in the New York Times [Markoff 1995] and coverage in the Wall Street Journal [Sandberg 1995b] and Daily Telegraph [Uhlig 1995].

Attempts to fix this problem introduced yet more problems. Under Windows the browser and server code, which appear to share the same random number code, don't close some of the file handles they use. While this has no serious effect on the client software (which doesn't run over extended periods of time), it does effect the server, which after a period of time runs out of file handles so that a number of calls related to gathering random data (some of them not apparently file-related) quietly fail, significantly weakening the random-number generation process [RingZero 1995]. The problem of insecure random number generation was not unique to Netscape, and has in the past beset XDM (which generates weak `xauth` keys), Netrek (a network game which generates guessable RSA private keys), an earlier version of the SecuDE security toolkit (which again generated guessable RSA keys), and Sesame (a European Kerberos clone) which uses it to generate DES keys.

Private Key Storage

Another flaw in Netscape concerns the way the server protects its private keys. The Netscape commerce server encrypts the key by hashing a user passphrase and using the result to key RC4, which is used to encrypt the RSA private key. The flaw found was similar to the one which makes Windows for Workgroups and Windows 95 passwords relatively easy to break [Gutmann 1995], both because an attacker can very easily try a large number of potential passphrases (in the message which explained the problem, a collection of more than 100MB of word lists was used), and because the private key contains around 100 bytes of constant, known data at the start. Because RC4 produces a fixed output for a given key, and because the plaintext being encrypted was known, the first 100 or so bytes of RC4 output could be recovered without knowing the encryption key and then used to decrypt anything else which had been encrypted with the same key [Gutmann 1996d]. Within a day of the details being posted, a successful attack on a commerce server key was announced by someone who used the insecure Windows NT FTP server to obtain a commerce server key and then broke the encryption using the code included in the original message which discussed the problem [Gutmann 1996e]. The consequences of this attack are rather serious, since it involves vastly less effort than breaking a 40-bit session key or factoring a 512-bit public key, yet once the private key has been recovered every session key it has ever protected in the past and will ever protect in the future have been compromised.

Java

Significant security holes are also opened up through the use of Java, which allows arbitrary programs downloaded from the net to run in a (supposedly) controlled environment on a host PC. By breaking out of this controlled environment, Java applets can act as trojan horse programs on the PC, bypassing normal security measures. The consequences of these security problems have been widely reported and include the destruction of data [Clark 1996a], the ability to access arbitrary files on the system [Felten 1996b], the ability to forward sensitive information to arbitrary machines on the net (bypassing firewalls and similar measures, since the "attack" comes from a trusted machine inside the security perimeter) [Williams 1996] [Markoff 1996b] [Martin 1996], or even run arbitrary native code on the machine [Felten 1996a] [Hopwood 1996] [Kennedy 1996]. This last class of flaws are the most serious, since they allow any code to be executed on a victims machine. The problem was in the class loading code for the browser and affected all then available browsers rather than just Netscape, and could be carried out simply by a victim viewing a web page containing the hostile Java applet. However the former flaw, the ability to connect to arbitrary hosts, also presents a problem since it can be used to penetrate a firewalled system from the inside. The machine running the Java applet can connect to other machines located inside the firewall (which often have very little or no security, since the firewall is supposed to provide all the necessary protection) and then forward the data back to the machine outside the firewall from which the applet originated.

Java problems can be combined with other attacks such as DNS spoofing (in which a fake address for a target machine is advertised), allowing a Java applet to connect to a normally disallowed target machine since the Java security manager thinks it is connecting to a safe system [Mueller 1996]. Other problems are less subtle, and can crash the browser (and, under some versions of Windows, the operating system as well) simply by connecting to a web page [Crash Netscape]. JavaScript, simple Java statements embedded in a web page which simplify the creation of web forms, presents problems of its own, not so much through any ability to damage a users machine like Java can, but because it can leak data from a users machine to the outside world. For example a JavaScript could trick the user into uploading data from their machine to any other machine on the Internet by presenting some innocuous button to the user which initiates the transfer. A list of further JavaScript problems can be found in [LoVerso].

Indirect attacks using Java are also possible. For example Microsoft provides a facility for its browser to automatically download new Java support code from Microsofts web site under the control of tags embedded in web pages [Java Support]. If an attacker knows of a security hole in a specific version of the browser, they can force the user to automatically up/downgrade to it and then attack their system through it. Another misuse of Java (which isn't a direct security breach) is to use the host system as a remote processor for a distributed computation such as encryption key breaking, as explained on page 17.

Java can also be used to create trojan horse programs which trick users into revealing confidential information such as passwords by displaying fake password-entry dialog boxes which are indistinguishable from the standard ones used by the browser (a somewhat limited demo version of the necessary code is available from [Password spoof]). There is no easy way to fix this problem, since

Java code which can draw to the screen to provide animation capabilities can also draw fake password entry dialogs, although the use of customised password dialogs can mitigate the problem to some extent [Tygar 1996].

One report comes to the conclusion that "because of the wonderful power of the Java language, security problems are likely to continue" [Neumann 1996a]. A detailed analysis of design flaws in Java itself is given in [Dean 1996], and Java security as a whole is covered in [McGraw 1997]. Because of these basic problems, even a theoretical "perfectly secure" version of Java (one with no implementation flaws) would still allow some types of attacks.

ActiveX

The types of problems described above are not unique to Java and JavaScript. Microsoft's ActiveX, which is a much more dangerous technology than Java since its applications have full access to a users system and can do anything which a native application can, has experienced even worse troubles. The Java "sandbox" security model can be likened to a ship with internal bulkheads which confine damage caused in one section to that section only. In contrast the ActiveX model removes these bulkheads, so that damage in one area will affect the entire ship.

A demonstration of the power of ActiveX came in early 1997, when the German Mitteldeutscher Rundfunk (MDR) TV station asked members of the Chaos Computer Club (whose earlier exploits are documented on page 27) to create a demonstration of how computers could be used to perform illegal electronic funds transfers — in effect, to steal money. The CCC members decided to use ActiveX in their demonstration, and the first version of the program was written in just 4 hours on 8th January 1997. After some tuning of the code, it was used on 15th January to transfer money from one account to another. On 27th January 1997 several press releases announcing the story were issued by media outlets, and the full story, including a demonstration of the program in action, was broadcast by the MDR on 28th January 1997 [MDR 1997].

The CCC's program functioned as follows. When the user viewed a "bait" web page, the ActiveX applet was automatically downloaded and installed on the users machine. It then checked to see whether the popular financial application Quicken was installed, and if it found it, it added a transfer order to Quicken's existing set of transfer orders. The next time the victim submitted their transfers (along with their PIN and TAN (Transaction Number)), the extra transaction was processed along with the existing ones, resulting in money being moved out of the victims account into any other account. There was no way to defend against this type of attack. Requiring a TAN for each transaction wouldn't help, because many businesses batched their transactions for use with a single TAN (transactions are often charged on a per-TAN basis), and even if more TANs were issued it would be impractical to require the entry of a separate TAN for each individual transaction.

Like waves in a pond, the story of the attack began to spread: the Berlin Tagesspiegel covered it on the 29th January [Tagesspiegel 1997], PC Week on 6th February [Seminerio 1997], the Dutch Daily Planet and c|net on 7th February [Daily Planet] [Wingfield 1997], Der Spiegel on the 8th February [Scriba 1997], the ClariNet newswire on the 12th February [ClariNet 1997], the New York Times and Ziff-Davis' Newswatch on the 13th February [Krieger 1997] [Ziff-Davis 1997], with further coverage in the following days (mostly derived from reports on newswires). As with other media coverage of similar events, some of the reports are rather garbled; the most accurate coverage was written by the authors of the program and appeared in the German iX computer magazine [Peter 1997]. After the iX article was published the code for the program was released by the authors, along with other programs such as one which turned off the AuthentiCode security checks, allowing other security-breaking applets to be downloaded and run without any checking.

It was suggested that Microsoft's Windows marketing slogan be changed to "Where does your money want to go today?".

This applet was the first example of what security experts had been warning about since ActiveX was announced: That the technology was inherently insecure, since it allowed arbitrary code to run on a machine and pretty much do whatever it pleased. The code could have made extremely expensive calls to 0900 numbers, or transmitted any file on the system to an arbitrary remote host, or installed software to allow a hacker to remotely control the machine, or planted further trojan horses or viruses, or any other of a vast array of malicious activities. Alternatively, they could elect to do something which, while not generally useful to a hacker, would be very noticeable to end users, reducing confidence in the computer system and software they are using. Examples of this are an application which takes

complete control of any NT server or workstation, rendering it useless for any other applications [PC Week 1996], and an application which shuts down the machine and even turns off the power on systems which support this functionality [McLain 1996].

Even supposedly safe ActiveX controls from well-known vendors can be easily misused to attack the machine on which they are running. For example Microsoft's own ActiveMovie control can crash Windows 95 by being asked to play a movie from the `file:///aux` URL (which corresponds to a DOS pseudo-device name). If the ActiveX control contains methods which write files to disk, they can be persuaded to overwrite critical system files which will open up security holes, crash the machine, or destroy all data on the hard drive. Several commercially available ActiveX controls can be exploited in this way to create a security hole/disk crash web page. One control even provides the ability to execute arbitrary system commands. The developers of the control stated that the control was perfectly secure provided that users were careful about which web pages they accessed! The inclusion of powerful scripting and macro execution capabilities (which in the past has lead to such things as Word and Excel macro viruses, several of which were distributed by Microsoft themselves) means that even if the author of an ActiveX control is honest and legitimate and goes through the appropriate Microsoft certification process, it doesn't take much effort for a less legitimate control to use the first one to do its dirty work. The Java sandbox model wouldn't allow this type of interaction, but with ActiveX the ability of controls to freely interact with any part of the system is regarded as a feature (which in fact it is, provided security isn't an issue) . If the developers of these controls have taken the step of obtaining AuthentiCode digital signatures (which several have), these controls will be automatically loaded and executed even with the highest security settings in Internet Explorer enabled [Smith 1996]. Since ActiveX is (currently) the key to Microsofts long-term strategy of eliminating the distinction between information stored on the desktop and information stored on a network, this problem is likely to grow in the future[6].

Unlike a Java applet, ActiveX isn't something tangible which is installed and run, but instead is an abstract concept which permeates an entire system. A security hole in Microsoft Word, or Excel, or Explorer, is also a potential security hold in ActiveX. Microsofts response to these issues has been to point to its AuthentiCode technology, in which developers digitally sign applications before they distribute them, and the web browser checks the signature before it installs and runs the application. However there are a number of problems with this approach, some due to end users or developers, and some due to the concept of AuthentiCode as a whole.

The problem at the end user level is that Windows users have been conditioned to click on "OK" for almost any dialog box which appears and many neophyte users will have no idea of the implications of an AuthentiCode warning, so that it is likely a malicious ActiveX application will succeed in getting permission to run on many systems, even with no AuthentiCode signature and the maximum security level enabled on the browser. Attackers can increase their chances of success by providing a number of apparently harmless and useful ActiveX applications, of which only one needs to obtain run permission in order to disable the checking of a future malicious ActiveX component. This needn't even be done deliberately — one company recently produced a web search utility which (accidentally) sidestepped Internet Explorers AuthentiCode facility and would have allowed any future software provided by the company to have run unchecked (the software has since been fixed) [Wingfield 1996b].

The problem at the developer level is that most developers see AuthentiCode as a user perception issue and not as a security issue. They will therefore continue to ship the same buggy, insecure software as before, but know that users will be more ready to accept it because of the flashy picture of a certificate which pops up before the ActiveX control is installed. At least one company has even gone so far as to apply their own signature to more than 1600 third-party ActiveX controls — which could contain code to do absolutely anything — because their end users liked it: "The cute 'certificate' dialog has a very positive impact on user confidence" [Pettitt 1997]. There have also been reports of corporate MIS departments signing third-party controls, again because this would make their users more likely to run them without question. In this case a malicious coder doesn't even have to sign their ActiveX control, because another company which is concerned about making a good impression on its users will do it for them! Once one of these cuckoos-egg controls, complete with its AuthentiCode signature, is loaded onto a system, it can proceed to do whatever it wants with the contents of the system, including

[6] It was suggested on one security mailing list that innovations like "ActiveX on the Web" deserved taglines such as "Building Tomorrow's Security Holes Today" or "The Insecurity Goes in Before the Name Goes On".

(because of the lack of a sandbox) subverting other controls from sources which do try to take care with their coding and signatures.

The overriding problem, however, is that the entire AuthentiCode concept is fundamentally flawed. AuthentiCode offers a only single, very fragile level of security, namely the initial check when a control is loaded, assuming the check hasn't been turned off by the user, or accidentally turned off by another application, or deliberately turned off by a malicious application, and that the application hasn't been conveniently signed by a third party, and that the user doesn't simply treat the warning like any of the countless other warnings which Windows will pop up during the course of the day.

Once the application has made it past the initial check, there is (contrary to Microsofts claims) no other security available. The reason for this is that any evidence required to provide accountability in case of an attack is stored on the machine which is under attack. Any attacker with the technical skills required to write a malicious ActiveX control should experience no trouble in removing any evidence of what caused the problem. More seriously, they might plant evidence pointing to some innocent third party, a technique which has been used extensively by hackers in the past to lead investigators on long wild goose chases.

Because of this, even if some sort of "evidence" is discovered on the system, there is no way it can be trusted since the malicious application had free run of the system during its attack. Even if there were some facility for recording which ActiveX controls were installed on a system in some tamperproof manner such as the Unix `syslog` facility (which is only possible in a networked environment with a secure host which performs the logging), there is still no real accountability. If a user powers up their machine one morning to find a blank hard drive, there is no way to tell what happened, or who or what was responsible — a software bug, a malicious user, a virus, or one of the 300 AuthentiCoded ActiveX controls which used to reside on the system. Expecting a malicious application to leave evidence of its wrongdoing on a system after it has completed its task would seem to be akin to expecting a burglar to leave a signed confession at the scene of the crime.

The problems extend beyond Java and ActiveX to other kinds of embedded executable content as well. For example the ability to embed macros in documents viewed and downloaded by a browser, in combination with security holes in the browser, allows an attacker to execute arbitrary code on a victims system whenever they view the attackers web page [Felten 1996c]. Although this was subsequently fixed, the solution was to issue warnings for all local files as well as remote downloads, so that after the first dozen or so messages the user was likely to simply automatically click "OK" whenever another warning popped up [Walsh 1996]. In addition a hostile application could access the Windows registry (the system-wide database of configuration options) to quietly disable the warnings. This creates a nice niche for "espionage-enabling" viruses which disable or patch various security features in operating systems or application software to allow later attacks. At least one security organisation already uses such a program, a modified stealth virus, for this purpose. Even if the hostile application can't hide itself from the user, all it needs to do is open the gates and let in a new, harmless version of itself (along with who knows what else) which will pass scrutiny for potential security holes.

CGI Scripts

Common Gateway Interface (CGI) scripts, which are used to enhance the capabilities of web servers, also pose a security problem. Because a CGI script becomes an intrinsic part of the web server, it must be written with the same care as the server itself. Server-side includes, fragments of code embedded in HTML documents, are another source of problems as they can be used to instruct the server to execute arbitrary system commands or CGI scripts. The fact that server-side includes are much easier to write than CGI scripts makes them that much more dangerous because they can be created even by inexperienced server administrators who have little knowledge of potential pitfalls.

CGI scripts can be abused in many ways. Since CGI scripts are often written in the interpreted Perl programming language, an attacker who can somehow obtain the CGI script (for example by taking advantage of an FTP server which allows any file on the machine to be retrieved or through a Java or JavaScript trojan horse) can analyse the source code for any flaws or programming errors. An attacker will typically use holes in CGI scripts to obtain ordinarily inaccessible information such as data files from a server, or to execute arbitrary commands which range from deleting files through to allowing an them to log onto the system. This attack usually involves tricking the program which interprets the CGI script into executing operating system commands through the clever use of command substitution, escape sequences, metacharacters, and other features (or bugs) of the command interpreter. This affects

not only Unix systems but any system with some form of command interpreter. For example several Windows-based servers exhibited bugs based on their handling of CGI scripts implemented as DOS batch files. Although the known bugs have now been fixed, the fixes assume that the batch files are being run in conjunction with the standard DOS command interpreter, and can be bypassed if the system contains an alternative, more powerful interpreter such as JP Software's 4dos, which allows sophisticated command aliasing, substitution, and use of metacharacters.

If the server also provides an FTP service, an attacker may be able to upload a CGI script to an FTP directory and then trick the HTTP server into executing it by requesting its URL (this can be avoided by forcing all scripts to reside in a single tightly-controlled cgi-bin directory). The ability to access a CGI script from anywhere (and not just the form it is associated with) also provides a powerful tool for an attacker, who can use this ability to directly feed the script input data which it may be unable to accommodate, or may handle in ways not intended by the scripts' author. Even a script created to serve as an example of safe CGI programming was recently shown to contain a flaw which allowed an attacker to execute arbitrary commands on a victims system [Stein]. Further security issues related to the use of CGI scripts may be found in [Phillips] and [CGI Security].

Other tricks may also be employed to trick servers (and in some cases clients) into executing arbitrary code. One method, which led to a series of embarrassing security holes in MSIE in March 1997, was to place Java code in a browsers cache and later execute it through a `file:` URL. Since the MSIE cache layout is transparent, this attack is easy to perform [Karas 1997] [McGraw 1997]. Several variants of this type of attack and similar attacks based on executable content were used to bypass a series of bugfixes to MSIE.

Other Problems

Cookie files can also represent a security problem. Cookies were designed to allow web sites to maintain state information such as user preferences across connections. This allows servers to store arbitrary (and often unknown) information on the users machine without the users knowledge. Some sites store large amounts of data in cookie files (Microsoft Network), or even leave account names and passwords neatly tagged and unencrypted in the file (Boardwatch) [Taylor 1996]. This means anyone with physical access to the machine can obtain all sorts of details on the user of the machine, and if they are in the habit of using the same password across multiple accounts, the cookie file can provide a goldmine of information.

Finally, a number of observers have urged for caution in the headlong rush to be the first to offer electronic commerce services on the Internet, pointing out that design and implementation defects in existing operating systems, network software, and application software products can often interact or interfere with other components to create security holes [Neumann 1996b]. For example although Windows NT itself is reasonably secure, a number of products either bypass this security as part of their normal operation (for example Webconsole [Cooper 1996], MS Internet Information Server [Baron 1996], [Lindstrom 1996], [Klaus 1996], `rsh` [LeBlanc 1996a], and the FTP server [LeBlanc 1996b]) or require security holes to be opened up in order to operate (for example [Ramos 1996] and the MS SQL Server [Merriman 1996]), making an otherwise reasonably secure system an easy target for attacks.

Liabilities of Weak Encryption/Poor Security

The susceptibility of a security system is usually defined in terms of a criminal or publicity attack. However by far the greatest threat comes from a legal attack in which the system is required to provide evidence in court. Many security systems are designed more to show due diligence or to provide a means of shifting blame onto others ("our systems are infallible, it must be your fault") than to provide solid evidence in court. The typical life cycle for a security product has been suggested as:

$$\text{Research} \Rightarrow \text{Development} \Rightarrow \text{Engineering} \Rightarrow \text{Product} \Rightarrow \text{Litigation}$$

after which the cycle repeats, although for a number of schemes currently being rushed into use on the Internet the model is probably closer to:

$$\text{Development} \Rightarrow \text{Product} \Rightarrow \text{Litigation}$$

with progress currently at the second stage.

The main problem which faces these systems is that cryptographic evidence is currently very easy to challenge in court. For example under UK law, the operator of a computer system must produce a statement showing that the system was working correctly at the time it was required to provide evidence, something which is extremely difficult to prove for any system. One tactic which has been used in UK court cases involving disputed ATM transactions is to request a complete set of the banks security and quality documentation including security policies and standards, key management procedures and logs, audit reports, test and bug reports and logs, and various other records. Since these frequently don't exist or are extremely difficult to produce, the case collapses when the evidence isn't forthcoming. In one trial in the UK the judge ruled that the defence had to have access to this information to determine whether anything could have made the computers fallible, and when the bank couldn't produce it ruled that all other computer-generated evidence presented by the bank was invalid [Anderson 1996a]. Unfortunately this tactic works whether the defendant is innocent or not.

A more serious problem is presented by organisations forced to use weak US-government-approved encryption for security. It is likely that the position on liability for Internet-based financial transactions will follow the model used for credit card transactions in which customers are liable for some small fraction of disputed transactions and (usually) merchants are responsible for the rest [McCullagh 1994], which may make merchants extremely reluctant to commit to an electronic commerce system whose ongoing (in)security has been widely publicised. An even more serious situation occurs in some countries which have legislation which makes company executives personally liable for fraud against the company if it is perceived that they have not taken adequate steps to prevent the fraud [Geary 1994]. Some countries also make professionals such as doctors and lawyers criminally liable for revealing confidential client data to outsiders, so that the use of security systems which are known to be inadequate might be interpreted by courts to be illegal in some countries [Datenschutzberater 1994]. In the highly litigious US the threat of legal action has encouraged firms to spend more money on information security than their non-US counterparts [Heinlein 1993]. The US Federal Sentencing Guidelines for Organizational Defendants, issued in 1991, contain clear, systematic rules for judges to follow when sentencing corporations found guilty in federal liability cases. This makes system administrators and managers, and their bosses, and their bosses bosses, personally liable for any negligent acts a company is convicted of, since they must show due diligence in their supervisory responsibilities (the law states that they must take all reasonable and prudent precautions to safeguard against theft of information. It is quite possible that courts will not regard the use of software whose poor security has been the subject of international media attention as fulfilling the requirements for "reasonable and prudent precautions"). To add insult to injury, the company shareholders could sue if the company stock price slips because of a security breach.

The impetus for providing strong security measures may come from unlikely sources. The Mosler Safe Company has reported that the driving force behind the design of strong safes and their deployment to banks and businesses was supplied by insurance companies, who offered rate discounts if stronger safes were installed. Until similar financial motivation exists for computer security measures, the main incentive to providing strong security has been the threat of adverse publicity.

The lack of insurance-based incentives is complicated by the fact that it is extremely difficult to provide any kind of assessment of the value of security measures. For example spending $10,000 on a firewall doesn't guarantee $10,000 worth of security — it may provide $100,000 worth, or none at all. One company which looked at the issue purely in financial terms calculated that if the average security breach costs $200,000, and there is a 10%/year chance of a breach, then spending $40,000 on installing and administering a firewall for a year would lose them $20,000/year over having no firewall present at all. This kind of maths makes it extremely difficult to convince managers and accountants to invest in security measures, and equally difficult to convince insurance companies to provide cover based on such safety measures. IBM, one of the few companies which provides insurance cover with its firewall product, can only do so because they are big enough to ignore the technicalities and can act as their own insurance company.

Systems used for electronic commerce may also need to be able to provide a number of detailed auditing functions. For example under US law a system may be required to provide detailed transaction information under subpoena, prevent disclosure to law enforcement except under subpoena, report any transactions over $10,000, keep a record of any transaction over $100 for at least five years, record any action on a joint account, provide copies of consumer records to the consumer, delete obsolete

information on the consumer, provide a record of personal income for taxation purposes, and so on ad nauseum [Camp 1995].

Designers of security standards have to date concentrated almost exclusively on technical attacks on the system, with little regard to legal challenges. In order to create systems which will withstand court challenges it will become necessary to involve lawyers in the design process along with cryptographers and security specialists. This precaution will help protect commercial organisations who are more used to paper-based security measures from falling into various traps based on their unfamiliarity with the security measures required for electronic commerce. Users must be assured that an EDI system provides them with the same or better protection against mistakes and fraud than the equivalent paper-and-signature based system they are used to. [Anderson 1994a] should be required reading for anyone considering implementing an electronic commerce system which will be required to withstand legal scrutiny.

Finally, basic ethical considerations for taking strong measures to guarantee the privacy and integrity of data should also be taken into account. Although information privacy concerns vary somewhat across cultural and national boundaries, it can be argued that information privacy should be treated as a "hypernorm", a principle which is valid across all cultures. Organisations therefore appear to have an obligation to protect information privacy regardless of distinctions in national levels of concern or regulatory approaches [Milberg 1995]. Some organisations already have guidelines covering these issues, for example the British Computer Society Code of Practice requires that all reasonable measures to protect confidential information should be taken [BCS 1995].

Digital Signatures and the Law

In some areas of electronic commerce the law is very gradually catching up with the technology. For example a few states in the US now recognise the legal status of digital signatures. Electronic commerce transactions constitute business contracts, which means they must have electronic signatures which have the same legal significance as paper signatures. Florida recognises digital signatures as an alternative form of a written signature without going into details of how an electronic document must be signed, and authorises the Secretary of State to act as a certification authority [Florida 1996]. Utah and Washington [Washington 1996] have similar laws, and Georgia [Georgia 1996], California and Virginia are currently considering digital signature legislation (some of these bills may already have been passed by the time you read this paper). Many of these laws are based on the American Bar Association digital signature guidelines [ABA 1995]. In other locations it may be necessary for both parties to sign a traditional written agreement in which they agree to use digital signatures, although whether this will be accepted by a court is still open to question [Brown 1993] [Brown 1994]. Other countries are also beginning to consider digital signature and electronic commerce laws [Kuner]. A good overview of the legal ramifications of electronic commerce is given in [Wright 1996a].

Unfortunately a lot of the work on digital signature legislation is disorganised, with little agreement on what constitutes a digital signature and how it is to be used. For example the California law is very generic and provides for any agreed-upon mark to be used as a digital signature. In contrast the Utah legislation is very specific about the use of asymmetric cryptosystems, CA's, certification revocation lists, and all the other trappings of X.509 (for this reason it is often referred to as "the law of X.509"). All security in this system is tied to the users private key, so that compromise of the key can lead to substantial difficulties. In addition the use of an X.509-based system may require lengthy (several pages long) legal disclaimers in certificates, making them non-machine-processable. The main problem with digital signatures is that there is currently nothing like this — a general-purpose mark which may be used to authenticate or authorise almost anything to anybody — in western culture, which will make for heavy going in getting it accepted by companies, financial institutions, and courts.

Even the specifics of digital signatures are hard to pin down. For example a merchant may not care whether an account specified in a signed transaction exists or not, all they care about is whether they will be paid. Therefore a certificate from a bank verifying that a certain party has an account with them is worthless to a merchant unless there exists an additional mechanism allowing the merchant to verify that the account actually contains the necessary funds to cover a transaction. In effect what is needed is not just a certificate of an account, but also a certificate that the account is in good standing, a criterion which cannot be met with a fixed, long-term certificate. For this reason there may be little incentive for a bank to take on the liabilities of issuing certificates if any transaction requires additional online verification anyway.

The uses of certificates for proof of identity are also open to some debate. If someone proposes to carry out a transaction with a merchant, the merchant will verify whether they think the person is a valid customer, not whether some distant authority considers them a customer. A centrally-issued certificate may serve as a simple proof of identity, but won't be sufficient for further use except in a few specialised situations like governmental or military contexts which have a well-defined concept of centralised identity and place in a hierarchy. In contrast in the world of electronic commerce almost all relationships are bilateral and involved a privately selected guarantor.

In a (perhaps unconscious) drift in the direction of certificates being general assertions about a party, the line between "certificate" and "signed message", in which a certificate becomes a series of assertions about the subject of the certificate, is eroding quickly. Companies like Netscape and protocols like SET are already using the `extensions` field in X.509 certificates to add ever-increasing numbers of assertions and statements about the subject of the certificate to the basic certificate structure. In some cases these extensions can extend to complex multipage legal disclaimers and other non-machine-processable items, which entirely defeats the point of X.509 certificates.

Setting up a certification authority (CA) is the absence of strong protection through legislation is currently a risky business. A CA is difficult to operate, the business model is inadequately developed, and the potential legal liabilities if problems arise can be substantial. The risk/benefit tradeoff in operating a CA is such that even large corporations have avoided the problem, or user alternative solutions which avoid the issue such as the centralised distribution scheme described on page 32.

There are all sorts of possible pitfalls awaiting the first parties involved in a court case which covers digital signatures. For example most digital signature schemes operate in the background with little intervention from the user. Because of this a court may rule a digital signature invalid because the user was not aware of what they were signing, or even that they were signing something, or a user may be able to have a contract voided by claiming unfamiliarity with the digital signature process. Until the legal situation is sorted out, it may be necessary to make the user aware of what it is they are signing, and that they are signing something which qualifies as a legal transaction. At least one security programming interface already contains some measures to implement this [Microsoft 1996]. An extreme case of this is the format used by the US Internal Revenue Service, who require signatures on documents not so much to verify the authenticity of the return, but to verify who filled out the return. This is to avoid the situation where taxpayers can claim that a fraudulent return was filled out by their accountant and not by them. For this reason electronic returns contain declarations of the form "…under penalty of perjury I declare…", making anyone who completes a fraudulent return criminally liable. The IRS uses a digitised signature at the bottom of the form which authenticates a tax return, with the wording "Below is the signature in which I agree to the above", followed by the signature, which is cryptographically tied to the 8453 tax form which it accompanies. The main purpose of this signature mechanism isn't to verify the contents of the return, but to uniquely identify an individual which the IRS can nail if the return is fraudulent [Wright 1996b].

An Analysis of Weaknesses and Potential Attacks

This section covers some of the main areas of concern when implementing a security system for use on the Internet. Although it would be possible to write an entire book on this topic, this section restricts itself to coverage of four major security areas whose exploitation has been detailed above.

Attacks on Session Keys

The most obvious attack on a session key is the possibility of applying massive computing resources to brute-force key recovery attacks on a 40-bit key. One way to speed up an attack is to use custom hardware, which can often perform the necessary operations far faster than an equivalent software implementation, especially for algorithms which were originally designed for hardware. A recent report [Goldberg 1996b] considers the feasibility of attacking RC4, A5 (the encryption used in GSM cellphones), DES, and CDMF (Commercial Data Masking Facility, an implementation of DES with a 40-bit key, which allows it to be exported [Johnson 1993]) based on actual implementations of key recovery engines using programmable hardware. A5, which was optimised for simple hardware implementation [Racal 1988], can be searched at the rate of 13 million keys/second for a US$100 investment (at May 1996 prices) in programmable hardware. Keys for DES (which is also optimised for hardware implementation, although significantly more complex than A5) can be searched at 550,000 keys/second at the same cost, so that a US$745 investment will provide the hardware to recover a 40-bit

CDMF key in one day, and a $15,000 investment will recover a key in an hour. RC4, on the other hand, was designed for efficient implementation in software, and exhausts the on-chip resources of the devices used to perform the attack, requiring the use of slow off-chip memory. In addition the algorithm really can't be implemented very well in hardware, so that a software search using a general-purpose computer is about twice as fast as a hardware implementation.

The possibility of breaking DES encryption should also be considered. Although the issue of DES key recovery hardware has been debated for almost as long as the standard has been around [Diffie 1977] [Hellman 1979], it wasn't until 1993 that complete constructional details for a DES-breaking machine were published [Wiener 1993] (this type of device has since become known as a Wiener machine after the author of the paper). This would recover a DES key in 3.5 hours for a US$1M investment in hardware, or 35 hours for the economy US$100,000 version which might be built by budget-conscious criminals. A typical target for such a device would be the zone control (interbank) keys on EFT networks, from which the investment in hardware could be quickly recouped. By 1996 the cost of constructing such a machine had dropped to approximately one third of the 1993 cost. For example one company offers quick-turnaround laser-programmed gate arrays of the complexity required in a Wiener machine for US$10 in 5000+ quantities, so that US$50,000 worth of chips clocked at 50 MHz could search 250 billion keys/second for a total search time of 40 hours for a DES key. Breaking a 40-bit key would require 2 seconds.

The Business Software Alliance, concerned about the threat to information systems through brute-force key recovery attacks, commissioned a study on key lengths for encryption algorithms in early 1996 which looked at the feasibility of attacking various algorithms and key lengths both in software and using custom hardware. The study concludes that "40-bit key lengths offer virtually no protection" and "even DES with 56-bit keys is increasingly inadequate" [Blaze 1996a]. Taking the estimated cost for recovering a single 40-bit RC4 key, amortised over the estimated 3-year lifetime of the hardware, results in an average cost of 8 cents per recovered key, which led some people to claim SSL had "8-cent encryption" (the assumptions for RC4 were probably somewhat optimistic since they didn't rely on an actual implementation. The DES estimates, however, were based on actual implementations).

With the increasing availability of fast PC's based on Pentium processors, software-based DES key recovery attacks have also been considered. Some of the possible tricks include eliminating the unnecessary initial and final permutations (the data being checked can be pre-permuted once before it is input to the DES transformation), combining the S-boxes and P-box operations, eliminating the 1st round by changing the way in which subkeys are tested for, eliminating the 16th round (the 15th round provides 32 bits of final output, allowing an early-termination test to be performed), working with the key in a pre-expanded state (so that a key expansion is only ever performed once), and using Pentium-optimised code which is tuned to take advantage of superscalar execution and cache management strategies [Trei 1996]. A P5/133 running this code can check approximately 600,000 DES keys/second, so that a modest network of 25 such machines (typically found in a university lab or small company) could recover a CDMF key in less than half a day on average[7].

In early 1997, the US computer security company RSA Data Security Incorporated initiated a competition to break algorithms with various key sizes, in particular DES [RSADSI 1997]. The smallest key used in the challenge, with a size of 40 bits, was recovered by a group at U.C. Berkeley within 3 ½ hours of the challenge being issued. The key was `F0 43 F1 81 31`, the search proceeded at around 100 million keys/second, and the recovered original message read "This is why you should use a longer key" [Goldberg 1997]. Only minutes later a European group announced the same result. The work was done mostly by university students, with most of the computing power coming from commodity UltraSparc and Pentium machines [Caronni 1997a].

The European group then decided to go on to attack the next keysize, 48 bits. The key of `74 A3 53 CC 0B 19` was recovered in 13 days after a total of 162,082,778,549,251 keys had been checked, with a

[7] Some consideration needs to be given to the environment in which the key recovery is performed. Under Windows 96 and NT, some multimedia applications — even if they appear to be idle — may unnecessarily poll sound devices even if they are turned off or disabled, resulting in a key recovery performance loss of up to 30%. It may be necessary to stop multimedia-related tasks in order to achieve maximum performance. In addition if the key recovery process is run as an idle task alongside a screen saver then it may never receive any CPU time at all, since screen savers will consume as much CPU time as they require and pass the rest on to idle tasks. Some screen savers, particularly the OpenGL ones, can consume 100% of the available CPU time so that other idle tasks never run at all.

peak rate of 440 million keys/second and an average rate of 140 million keys/second. Using the computing power applied to the 48-bit effort would have broken the earlier 40-bit key in 45 minutes [Caronni 1997b]. Like the much earlier attacks in RC4, a few glitches meant that the full computing resources which were available, which could have resulted in the time being halved, weren't used [Baechle 1997].

Recently work has been underway to develop a distributed keysearch protocol for use over the Internet. The protocol has two phases, the first of which is the request phase in which the client reports to a server its capabilities (such as available algorithms and how many keys it wishes to check), to which the server responds with an allocation message which informs the client that it can proceed, and supplies information such as plaintext/ciphertext pairs required for the keysearch.

Recently work has been underway to develop a distributed keysearch protocol for use over the Internet. The protocol has two phases, the first of which is the request phase in which the client reports to a server its capabilities (such as available algorithms and how many keys it wishes to check), to which the server responds with an allocation message which informs the client that it can proceed, and supplies information such as plaintext/ciphertext pairs required for the keysearch.

Other work in distributed computing will make encryption-breaking of this form much easier in the future. One example of this form of distributed computing uses Java applets embedded in web pages which are automatically downloaded and executed by anyone viewing the web page. An upcoming BBC Tomorrows World broadcast will demonstrate an example of this which uses the BBC's web server (which gets tens of thousands of hits a day) to distribute the Java applets. Although Java is up to 100 times slower than the C or assembly code used for the previous encryption-breaking events, the fact that it will be running on vast numbers of machines (unknown to the owners of the machines) should compensate for this.

Even faster key searches are possible by resorting to unusual ways of performing the DES encryption. By turning DES on its side and treating it as 64 identical 1-bit operations, a 64-bit CPU can be used as a single-instruction multiple-data (SIMD) parallel computer which performs 64 simultaneous 1-bit operations, with the 64 bits of each DES block spread over 64 words. In this way the expensive expansion and bit-permutation operations required for DES can be done at zero cost by changing the order in which the words (which correspond to 64 separate 1-bit quantities) are addressed. A 300 MHz Alpha CPU running 64 simultaneous DES operations in this manner provides a throughput of 17 MB/s for single DES or 6 MB/s for triple DES [Biham 1997]. This comes to about 2.5M DES keys/second for keysearching. A single computer could recover a CDMF key in 2 ½ days; 10 such machines could do it in 6 hours. A machine which Cray computer developed for the NSA could break the key in around 4 minutes; another one at Sandia National Labs could do it in 2 minutes. Note that these figures are for general-purpose computers which are not especially suited to this task, a specially-designed hardware implementation would be much faster than this.

The effects of such an attack vary depending on the attack type. A criminal attack on a 40-bit key probably isn't really viable, but a publicity or legal attack is. In addition it is not unlikely that within the next few years an attack on 56-bit DES keys might be funded by an organisation fielding a DES replacement which they would like to see adopted. Although an equivalent criminal attack would still be uneconomical, any public confidence in DES would probably be destroyed by the ensuing publicity. For this reason it is recommended that an accepted algorithm using at least 100 bits of key be used for bulk data encryption. Triple DES with its 112 or 168 bits of key is a safe bet, and although it is often criticised because of its slow speed a reasonably optimised implementation will still run at 650 kB/s on an entry-level P5/100, 800 kB/s on a commodity P5/133, and up to 1.3 MB/s on a high-end P6/200, well above the approximately 3 kB/s of a typical modem. For comparison, a direct translation of this code into Java ran approximately 60 times slower than the C original, making it unusable for practical purposes. A Java version compiled with a JIT compiler ran around 4 times slower than the C version which is probably acceptable for a bulk encryption algorithm like triple DES but still too slow for public-key encryption. Other algorithms provide an even larger speed margin, ranging from RC2 (up to 1024 bits of key) and IDEA (128 bits of key) which run at about the same speed as DES, up to Blowfish (448 bits of key) which runs 2½ times as fast and RC4 (up to 256 bytes of key) which runs 3 times as fast (6.5 MB/s on a P5/100 or 14.4 MB/s on a P6/200), however use of the latter is not generally recommended since the fact that it functions as a keystream-generator means that its use requires special precautions which are easy to get wrong for inexperience implementors.

Attacks on Public/Private Keys

Although a successful attack on a public key which would allow recovery of all session keys negotiated using the public key and the forging of digital signatures created with the private key is far more serious than an attack on a single session key, it seems unlikely that such an attack will be viable. The largest (publicised) application of computing power to breaking a single public key was the factoring of RSA-129, a 428-bit number, in 1993, which involved approximately 5000 MIPS-years of computing time over a period of 8 months donated by 600 volunteers to break a challenge set in 1977 [Atkins 1994] [Uehling 1994]. In 1996 RSA-130, a 432-bit number, was factored with improved algorithms so that only $^{1}/_{10}^{th}$ the computing power of the RSA-129 challenge was needed [Lenstra 1996]. Both efforts required considerable specialised resources, with the latter using 68 hours of CPU time and 700 MB of memory on a Cray C90 supercomputer in the final processing stages. Like attacks on session keys, attacks on public keys can be sped up through the use of custom hardware (for example an add-on card for an AT-class PC capable of giving it a multiprecision math performance greater than a four-processor Cray XMP cost about $4,500 in 1992 [Dubner 1992]), but the use of this type of hardware to attack public keys seems unlikely outside government-funded intelligence agencies.

The only known attack on a real public key was the breaking of the 384-bit Blacknet key in 1995. Blacknet was a cryptographically-secured information-trading network created as a hoax in 1993/1994, and used the weakest security grade of key offered by PGP. The key was broken in secret using about 400 MIPS-years of computation to demonstrate that a few determined individuals with relatively modest resources[8] could break a lower-grade key [Leyland 1995]. Similar resources could be applied to a key recovery attack on the 400-bit public keys used by the exportable version of Lotus Notes version 3, making it unnecessary to break the accompanying 40-bit session keys.

Neither of the previous attacks are currently feasible except as publicity attacks, since the US government allows the general export of software using 512-bit public keys. These are probably safe from general attack for a few more years, although organisations with less modest resources than those used for the Blacknet attack can almost certainly break these keys right now, with a recent paper showing that it could be done in 3 months using general-purpose Pentium-class computers [Cowrie 1996]. The SSL version 3 protocol definition recommends that anyone contemplating using 512-bit public keys for electronic commerce transactions change the key daily or after 500 transactions, more frequently if possible. RSA Labs also recommend against the use of 512-bit keys, in addition to recommending that keys (regardless of length) be changed every few years to minimise the vulnerability of any one key. Another expert opinion holds that 768-bit and 1024-bit RSA keys will fall within the next 10 and 20 years respectively [Odlyzko 1996].

Increases in computing power over time can make public-key encryption stronger (rather than weaker) if managed properly by increasing the key size periodically as computing power increases. The reason for this is that an improvement in computing power which allows an attacker to factor a number one or two digits longer will allow the use of a key one or two dozen digits longer. Adding a reasonable safety margin when choosing a key should eliminate any concerns over the gradual erosion of its security over time.

The principal concern with a key recovery attack on a public key rather than a session key is that the public key will be used to establish a large number of session keys, making the security of the public key far more critical than that of the session key (there are some key exchange protocols which avoid this weakness, for example [Diffie 1992]). If the public key is used in an electronic payment system, an attacker obtaining the private key is effectively given a blank cheque which can't be stopped and which, until the account is drained, has no withdrawal limits. Unlike forged paper money, which can generally be identified under close scrutiny and traced back to its origins, a forged payment message is indistinguishable from the real thing. As a result there is less incentive for people handling the payment to check its authenticity, because it'll still be accepted by the bank (provided there is money left in the account) because it too can't tell that it is forged. This is even more dangerous with systems like Mondex which rely on the physical security of the card containing the digital cash, because the only way to identify that fake Mondex cash is in circulation is when inflation starts rising for no immediately obvious reason, at which point confidence in the banking system is likely to be shaky.

[8] It is possible that the Bellcore MasPar which contributed about 50% of the computation doesn't really qualify as a "modest resource".

For this reason it is recommended that a public key of at least 1024 bits be used, and the key size be periodically reviewed and increased if necessary as further developments in factoring take place. Legal requirements that documents be retained for a number of years may mandate the use of keys of at least 1024 bits in order for a digital signature to remain valid for the expected lifetime of a document. A 1024-bit RSA encrypt/signature check operation takes 2 ms on a commodity P5/100, and a 2048-bit encrypt/signature check takes 7 ms. The corresponding RSA decrypt/signature creation operation takes 0.1 s on the same system, and a 2048-bit decrypt/signature creation takes 0.8 s (these drop to 0.04 s and 0.3 s respectively on a high-end P6/200). There is no noticeable delay involved in using 1024-bit keys, and even 2048-bit keys create only a minor overhead which is insignificant compared to other issues like network delays.

Attacks on Software Distribution Channels

By attacking online software distribution channels, it is possible to substitute weakened forms of the encryption algorithms in the software which can be exploited by an attacker. For example an attacker could watch data flowing across a network and substitute a slightly modified web browser which chooses its encryption keys in a predictable (to the attacker) manner rather than choosing them randomly. Such an attack has already been performed by patching 4 bytes of the session-key-generation code in copies of Netscape loaded over NFS to generate a predictable key known to the attacker. This attack can easily be modified to compromise binaries during download by patching HTTP or FTP transfers. The software to implement this attack was created in under a day [Gauthier 1995]. By spoofing traffic coming from the distribution site for a popular package such as a web browser, anyone with direct network access on a machine between the FTP server and the main external Internet link (for example a student on one of the university Netscape mirror sites) could patch each copy of the browser which was retrieved from this site. Although some organisations won't allow software mirror sites for this reason, all it does is relocate the target of an attack from the mirror site to routers and systems near the central distribution site, and creates a single critical point of failure at this location.

Software to perform this kind of active modification attack can be easily created. Books like the TCP/IP Illustrated series [Stevens 1994] [Wright 1995] [Stevens 1996] provide an extremely thorough and complete guide to the entire TCP/IP protocol suite, including operating-system-specific details and special features, implementation details including complete source code, and information on tools such as the BSD/Berkeley Packet Filters which allow arbitrary manipulation of Internet traffic. There also exist programs such as NetWatcher [Netwatcher 1996], which allow real-time monitoring and logging of traffic and hijacking of sessions, so that an attacker can seize control of a session from a victim. These attacks can be combined with known bugs in browser software such as an implementation error in the Netscape Commerce Server in which the entire mater key for a "secure" connection can be sent in the clear. Recall from the previous discussion that the export version of Netscape generates 128-bit keys but sends 88 bits unencrypted to comply with US export restrictions. Because of an implementation error the server will accept connections with all 128 bits sent unencrypted, and both client and server will treat the connection as secure, for both export versions (with 40-bit effective keys) and non-export versions (with 128-bit keys) [Roos 1995]. A simple patch to the Netscape client as it is downloaded is enough to exploit this flaw, with the only way to detect it being to intercept and laboriously pick apart the data in the SSL traffic to ensure that it complies with the SSL specification.

This type of attack can be foiled if an initial secure distribution channel can be set up and then used to bootstrap an online distribution system. For example the initial security system might be distributed to the end user via offline channels on a write-protected floppy disk or CDROM which also contains software to verify the integrity of any future online software updates. Before an updated version of the software is installed, the original version can verify its integrity to ensure that the online distribution channel hasn't been compromised.

Trojan Horse Attacks

Getting to the keys for an encryption system indirectly saves an attacker the effort involved in trying to break the encryption itself. With the advent of technology such as Java and ActiveX, it has become possible to create trojan horse programs with the capability of siphoning encryption keys off a system and transmitting them to a remote machine over the Internet (despite strong theoretical arguments that it is possible to create a secure Java environment, there is a considerable body of experience showing that in practice it cannot currently be done by mere mortals). A program which takes passwords for all resources protected by the Windows for Workgroups login password already exists (in a deliberately

crippled form to make it useless as a hacking tool) [InfoWorld 1996]. A similar program to steal passwords protected by the Windows 95 login password also exists, although not as a trojan horse. Another trojan horse which steals credit card numbers as they are typed was developed as a publicity attack by a company providing an Internet transaction service which neatly sidesteps the problem by avoiding the use of credit card numbers [Pizzo 1996]. Password-stealing trojans have long been available for DOS (for example [DEPL], [KBDC], [KeyCopy], [KeyTrap], [Phantom]), or even as firmware "upgrades" [Harriman 1990]), and have recently appeared for Windows 95 [NewAPI32] (which actually allows the interception of *any* system call and not just keystrokes), [Password Thief], [Surveillance Agent]. It is not unforseeable that in the future further programs which add supplemental functionality to the keyboard driver or use similar techniques will appear. Even Windows NT is not immune, with the recent appearance of a device driver which has the (benign) effect of re-mapping keys on the keyboard, but which could easily be modified to capture those keys to a file or transmit them over a network. There is also another way to capture passwords under NT — let the operating system do it for you. The registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Lsa\Notification Packages` points to a dynamic link library (DLL) which captures all password changes to a file, allowing an attacker to collect all machine passwords for later use. By placing the DLL on a domain controller, the passwords for an entire domain (which typically corresponds to a company or corporate division) can be captured. This driver will catch and filter keystrokes before NT sees them, making it possible to capture any kind of information entered through the keyboard [Ctrl2Cap].

Even the supposedly protected environment of Windows NT isn't safe against a trojan horse attack. By taking advantage of the debugging facilities built into the Win32 programming interface, a hostile program can control and intercept any function calls made to security modules and obtain encryption keys and other sensitive data passing across the interface. The first step in intercepting these calls is to determine where the calls to the functions of interest are being made from the program being monitored (the "victim") to the security module (the "target"). This is relatively easy because of the way in which Win32 programs and dynamic link libraries (DLL's) link to each other. When a function in the target is called, the call doesn't go directly to the other module but instead uses one level of indirection through a jump instruction which is shared by all calls in the victim to the function in the target:

```
;         CryptAcquireContext( ... );
000010D6  CALL 0002C104
  [...]
;         CryptAcquireContext( ... );
00008C92  CALL 0002C104
  [...]
;         CryptAcquireContext( ... );
0001A204  CALL 0002C104
  [...]
0002C104  JMP DWORD PTR [00302C11C4]
```

The `DWORD` containing the functions address is found in the victims `.idata` (import data) section, which is set up by the Win32 loader [Pietrek 1994a]. In order to intercept the calls made to the target, a program merely needs to locate the appropriate `DWORD`'s in the `.idata` section of the victim and overwrite them with the address of its own interception routine. This requires loading the interception module into the victims address space, which can be done in a number of ways [Richter 1994]. The easiest way is to use the Win32 `WaitForDebugEvent()` and `ContinueDebugEvent()` functions to intercept the victim being loaded into memory. The monitoring program sits in a loop processing debug event types until it sees an `EXCEPTION_DEBUG_EVENT` with an exception status of `STATUS_BREAKPOINT`, which indicates that a new process is being started. The victim is then frozen before it can begin execution, and modified to call `LoadLibrary()` to load the monitoring program, contained in a DLL, into its own address space. Once unfrozen, the victim loads the monitoring DLL and generates another `STATUS_BREAKPOINT` exception which causes it to be re-frozen so that it can be restored to its original state before it is allowed to continue. No traces of the surreptitious loading of the interception module remain after this process has completed [Pietrek 1994b]. All that remains is to patch the victims jump points so that they take a small detour through the monitoring program on their way to the target. This can be done by walking through the `.idata` section and changing each jump point to jump to the monitoring program, and changing the corresponding location in the monitoring program to continue on to the target.

This form of interception can be used not only to recover keys and other sensitive information but also to perform active attacks in which the functionality of a security module is altered to suit the monitoring programs requirements. For example the following call to verify the integrity of a security module:

```
CPVerifySignature( hHash, signature, length, hPubKey, description,
                   0 );
```

could be intercepted and given the functionality:

```
if( description = rogue security module name )
  return( TRUE );
else
  CPVerifySignature( hHash, signature, length, hPubKey,
                     description, 0 );
```

thereby bypassing any checking on which security modules a system will and won't load.

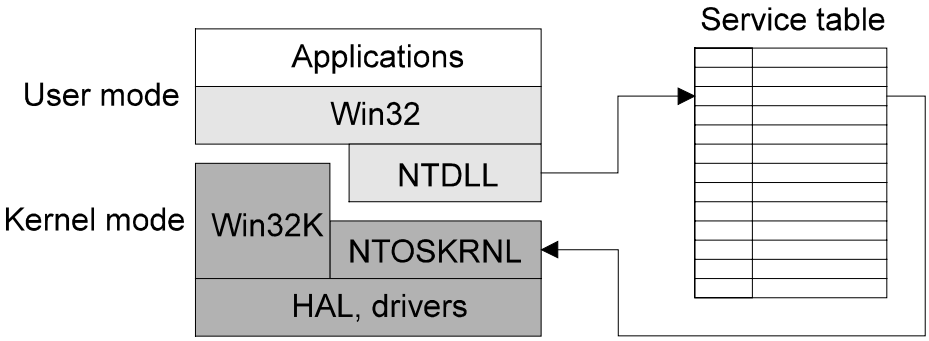
Another method for intercepting information is provided by so-called spy programs, which are usually used for debugging software which handles low-level system data and messages. Under NT messages and data from any device or virtual device can be intercepted through the use of a filter driver, which are inserted between two layers of existing drivers and communicate with them through three special interfaces, two of which communicate with the driver layer above and below them, and the third which provides an channel to the outside world, typically to a monitoring program, as well as providing the ability to control the driver.

The first two interfaces communicate by means of NT I/O request packets (IRP's) which represent any form of I/O request to and from a device. These IRP's are passed up and down the layers of drivers, which may modified the requests or even create additional IRP's in the process. In addition each driver in the chain may process the request asynchronously, calling an I/O completion routine provided by the next driver up in the chain when it has finished. This provides an extremely powerful mechanism for intercepting information from any kind of device.

When the driver is loaded, the OS calls its `DriverEntry()` routine, which calls `IoCreateDevice()` to create the three interfaces described above, informs the OS which requests it wants to service (a filter driver should service all request types), and then inserts itself to the driver chain using `IoAttachDevice()`. From then on it simply passes all requests it receives to the next driver down in the chain using `IoCallDriver()`. In this way a spy driver can intercept all data moving to and from a device.

Like equivalent programs which provide this ability for DOS, it is possible to load and unload NT drivers on the fly by using the NT service control manager, which means a spy program can be transparently installed and removed at any point [Schreiber 1997].

A third way to intercept information involves intercepting calls to the Windows NT kernel `NTOSKERNL.EXE`, allowing data passed to system calls to be monitored both before and after it's processed. Windows NT consists of a series of layers, going from the Win32 level which is usually seen by programmers, down through the NTOSKRNL level, which does most of the work, and eventually to the HAL (hardware abstraction layer) which handles the various hardware devices. All that the Win32 layer does is perform a bit of housekeeping such as parameter checking and breaking a request into one or more subrequests, before passing the kernel-specific parts of the request down to NTOSKRNL via another module, NTDLL, which handles the transition and mapping of calls from user mode to kernel mode. It is this architecture which allows other user-level subsystems such as OS/2 and Posix to be layered on top of NTOSKRNL. The layered architecture is shown below (the WIN32K module is the Win32 graphics engine which used to follow the same rules as the rest of Win32 but was moved into the kernel in NT 4.0 to boost graphics performance):



As an example, a call to the Win32 `CreateFile()` function would result in the Win32 subsystem making one or more calls to NTOSKRNL via NTDLL, the exact nature of which would depend on the parameters which were passed to `CreateFile()`. Typically the end result would be a call to `ZwCreateFile()`, the kernel routine which handles most of the work of `CreateFile()` (the '`Zw`' routines also have aliases so that they can be called with an '`Nt`' prefix if desired The '`Nt`' aliases will be used here).

There is nothing special about the NTDLL calls which require them to be called through the Win32 subsystem, and an application can call the '`Nt`' calls in NTDLL directly without having to go through Win32.

The mapping from user- to kernel-mode calls is performed using a table pointed to by a value in a processes Thread Environment Block (TEB), a data structure which contains all the information required to handle each process or thread executing in the system. This is shown in the diagram above. When NTDLL handles a system call, it converts any subrequests to system trap calls which switch to kernel mode, and the rest of the call is then handled by the kernel. The exact mechanism is shown below:

Win32 (user mode)

1. Perform parameter checking and general housekeeping.
2. Break request up into subrequests if necessary.
3. Call appropriate function(s) in NTDLL.

NTDLL (user mode)

1. Load eax register rwith service number corresponding to required service.
2. Load other registers as required.
3. Execute int 2Eh to switch to kernel mode.

NTOSKRNL (kernel mode)

1. Look up kernel service in service table using value in eax.
2. Call this service.

In order to be able to intercept these system calls, we need to either intercept the calls as they go from Win32 to NDTLL, or as they go from NTDLL to NTOSKRNL. As it turns out, intercepting them during the jump into kernel mode is relatively easy because of the way the service number is mapped to the required kernel service. Recall that the table used to perform this mapping is pointed to by a value in the TEB. Although this table could in theory be local to each TEB, it is in fact a system-wide table which is shared by all threads. Therefore in order for a process to redirect all calls made to a system service to some other piece of code, all that is required is for the process to read the address of the service table from its own TEB and modify the table entry corresponding to the service which is being redirected. Since the service table is shared by all threads, everything will automatically be redirected to the new code [Russinovich 1997].

This method of intercepting system calls is extremely powerful, since it is so easy to implement — a standard program can launch the device driver which patches the service table for the services of interest, and from then on all data going to and from things like password entry dialogs is filtered through the interception program, which can process it as required (for example it could forward it to an external host).

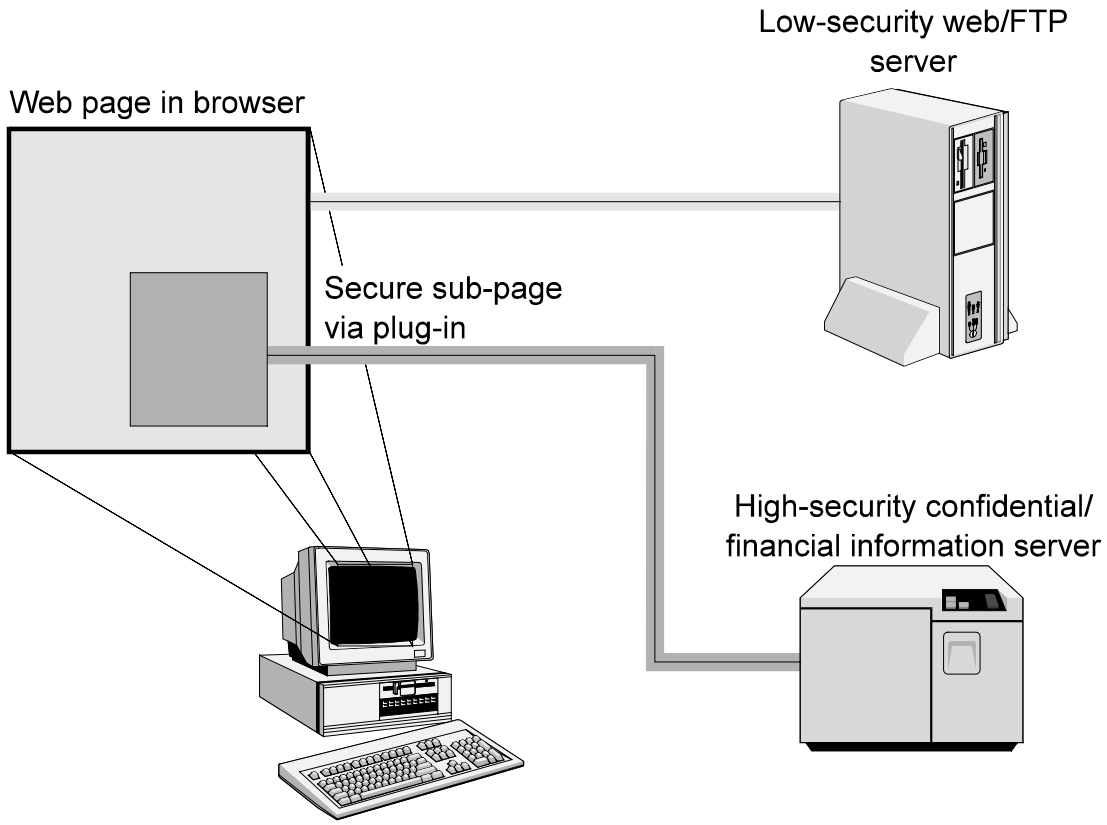
By using a Java or ActiveX program to either steal keys or implant weaknesses or monitoring programs in a system which bypass major security features, an attacker can save themselves the effort of attacking the security system itself. Although the previously mentioned exploits with Java and ActiveX have been simple demonstrations of what is possible and the people who carried them out have gone public afterwards, there is a strong incentive for an attacker to develop programs for more sinister ends and make sure they are never discovered (a program which patches a standard system library to provide only illusory security and then quietly deletes itself after it is first executed would prove almost impossible to detect).

There are relatively simple workarounds to this problem. In a corporate environment, a firewall can be configured to block any MIME data of type `application/binary` (the data type used for Java) while the hypothesis that Java is safe is subject to a little more practical testing. In an environment in which the client is directly connected to the server without going through a firewall, a server which requires a secure connection can send a Java or ActiveX application to the client machine which sends a "ping" message to the server. If the server receives the ping, it knows the client has Java or ActiveX enabled and can request that they disable it on their system before the secure communication session can proceed.

Fixing Insecure Systems

The many problems inherent in insecure Internet software, both due to US export restrictions and programming errors, can effectively be bypassed by utilising the ability of programs like web browsers to support plug-in modules which can provide the necessary security. The following discussion covers Netscape plug-ins as this is currently the most widespread browser, although Microsoft's ActiveX controls can be used in the same manner (the main difference is that dangerous ActiveX extensions can be installed automatically, often without the users knowledge, whereas plugins require active user intervention in order to be installed).

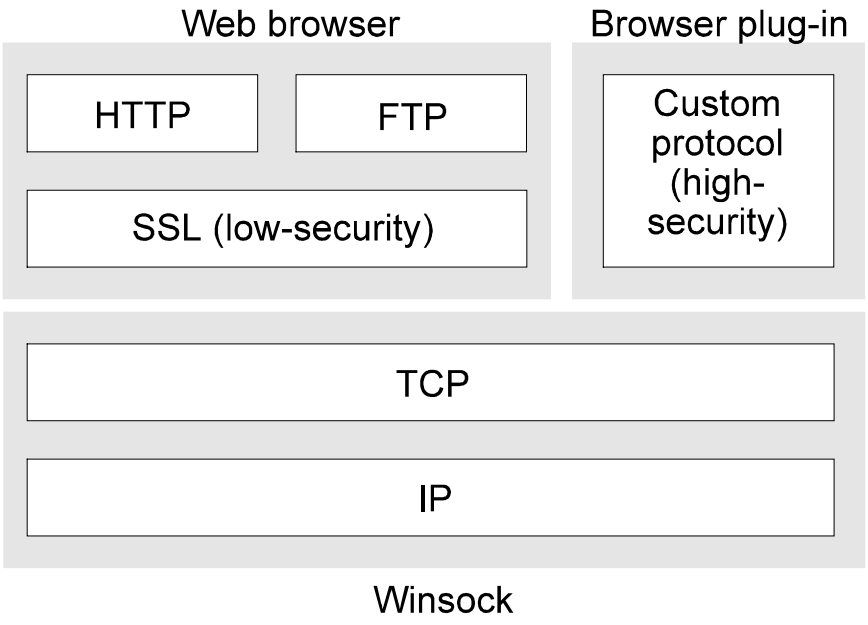
Plug-in modules were designed to provide seamless support for new data types such as video animation and sound. They can draw into and receive input as part of a larger HTML document, making them appear as an integrated part of a web page. In this manner the low-security sections of a document can be transmitted via a normal HTTP or SSL session, while the sensitive portions can be sent through a link managed by the plug-in:



Plug-ins are associated with a MIME data type for which the browser has no native support. When this data type is encountered, the browser looks for the application associated with it, loads it, and activates the plug-in. The plug-in then establishes a secure connection to the remote server and carries out a dialogue with the user through the subpart of the main browser window which is allocated for its use. Ideally the plug-in should be as independent of the browser program as possible in order to avoid being caught by security flaws and bugs in the browser. In the above diagram the plug-in uses its own connection, which is independent of the web browser, to transfer data in a secure manner, and makes it available to the user in a subsection of the browser window[9]. It therefore functions as a kind of "secure symbiote" which relies on the browser to start it up, but then carries on independently of the browser.

The separation of the main browser and the symbiote is critical. Although the browser will typically provide a number of services such as cacheing of data and a stream interface for network access, the symbiote should manage its own network connections and data explicitly — the data which the browser helpfully saves to disk could be critical information which can't be allowed to be saved to a persistent storage medium. Similarly, the remote server which the symbiote connects to should be on a physically separate machine on which it is the sole service. All other network services and accessibility on this server should be disabled to allow only the symbiote to connect, and only in a highly controlled manner. In this way someone finding a security hole in one part of the server (such as the various holes in the MS Internet Information Server, or using holes in the Windows NT FTP server to compromise the Netscape commerce server, see above) can't compromise the entire site.

The protocol layers used to implement the browser and symbiote interoperate as follows:



An alternative method for providing a secure link is to add a second level of security over the top of the low-security SSL layer. This approach is rather difficult since it requires the security system to have hooks into the browser it is being run on. Fortunately the browser itself provides these hooks in the form of Java. At least one vendor has taken this path and created Java modules which provide 1024-bit public-key encryption and IDEA or triple DES conventional encryption on top of the existing SSL protocol. The extra layer of security is implemented as client and server-side Java modules which run a performance-optimised variant of SSL called SRT (Secure Request Technology) over the top of the existing low-security SSL connection. The system attempts to avoid the security problems inherent in Java by digitally signing Java applets and providing browser plug-ins which check the Java applets and Java runtime environment for potential problems [X*Presso]. However the use of Java and weak 40-bit encryption to bootstrap a secure electronic commerce system raised serious enough security concerns with the Zentraler Kreditausschluß (central standardisation committee of the German banks, who had

[9] This system isn't perfect — OS/2 Warp has system-wide network socket descriptors, which means any other active process on the system can access a given programs network data — but it's a good step in the right direction.

standardised on pure triple DES and 768-bit[10] RSA encryption), that they didn't consider it for their online banking requirements.

All noncritical data such as informational text and graphics may be communicated over an insecure or low-security connection such as the one provided by a US government-approved SSL implementation. Any security-critical data should be communicated over the high-security connection provided by the symbiote or proxy using the strongest security possible. Strong encryption costs no more to implement than weak encryption, so it makes sense to use the strongest algorithms available. There is no need to provide different levels of security for different applications, with complex cost/benefit tradeoffs, since a conventional algorithm with a 64-bit key will run no slower or faster than the same algorithm with a 128-bit key (the same is not true for public-key algorithms, but since these are generally used only once in a session to exchange a session key or provide authentication, their speed isn't so important). Strong encryption will give users a greater sense of security even if it isn't strictly necessary (the advertising for a number of existing security products already emphasises their large key sizes to play on "bigger is better" expectations in users), and means that organisations such as banks and clearing houses which really do require strong security can use the same systems as everyone else, resulting in a cost savings as the same code base can be used by all parties. Everyone can use the *greatest* common denominator for their security. In addition, when selling computer security to users, perception is everything: There is no point in trying to persuade a customer that the fact that they saw the words "MD5" and "trouble" in the same sentence in the paper this morning doesn't make your product insecure; it's far easier simply to switch to SHA1 and issue a press release telling everyone how much more secure your product is than the competitions. Using a very strong (even unnecessarily strong) algorithm in the first place will avoid this kind of problem.

An additional reason for using the strongest possible security at the outset is that a system which is improved incrementally will provide a fertile training ground for attackers who can practice on the weak initially fielded system and then slowly work their way up to more sophisticated attacks as the security is improved over time. This technique proved very successful (and in many cases lucrative) for European pay-TV hackers, who over the years have progressed from simple techniques such as blocking card disable signals and fiddling with supply voltages in order to erase security fuses through to sophisticated attacks involving multimillion-dollar equipment such as electron-beam testers and focused ion beam workstations against which it is uneconomical to protect mass-market smart cards and microcontrollers. In the cases where the hackers were motivated by the profit motive rather than simple curiosity, revenues gained from one level of attack were used to finance the next round of attacks. Because the pay-TV companies didn't bother to implement strong security measures initially, they helped finance the development of better and better attacks, to a point where it is no longer economically possible to protect a system against determined attackers [Anderson 1996b].

Once the data has safely arrived at the server, there still remains the problem of getting it to the merchant. Although the number of businesses with direct Internet connections is growing every day, many smaller business don't have the expertise or staff to run their own servers and instead rent space on a system run by an ISP. This creates a problem when data needs to be securely transmitted from the ISP to the merchant. Even if the merchant has their own server, they still need to communicate with the clearing house, and although this was traditionally done over X.25 or dedicated PSTN links the attractiveness of an Internet-based solution means this functionality may increasingly migrate to Internet links in the future. SET is the first protocol to specifically address this problem, providing security not only from the client to the server but also to and from the merchant and clearing house. Unfortunately SET has a number of limitations (some of which are covered on page 33) which make it unsuited for general-purpose use. In many cases the problem doesn't exist — applications like online banking and interactions with larger merchants who have dedicated X.25 links to clearing houses don't require the data to be sent over the Internet a second time.

In the cases where a merchant relies entirely on the Internet for communications, care must be taken to ensure that both the front door (client to server) and back door (server to merchant or clearing house) are secured. A typical solution might be to use PGP or S/MIME to encrypt the results of the client/server transaction and forward the results to the merchant or clearing house. The problem is complicated by the fact that the ISP is probably running Unix on their servers, whereas the merchant would typically run some variant of Windows, requiring cross-platform encryption software. So far the most universal solution is to use PGP, which can be run in batch mode under both Unix and Windows

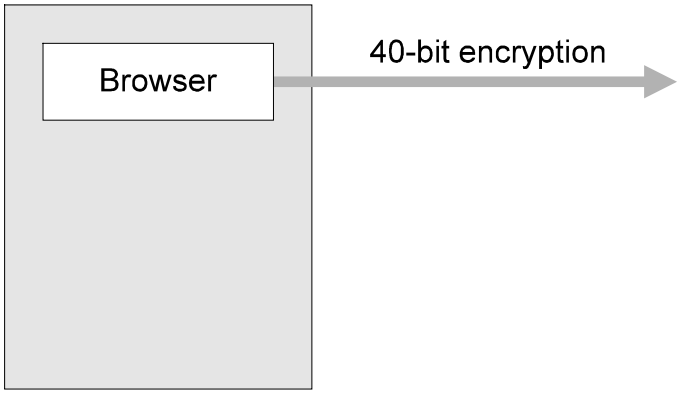
[10] This short key length was due to initial speed concerns, the key size is to be raised in the near future.

to encrypt and decrypt email being moved over an Internet link. S/MIME solutions are mainly available as part of interactive Windows software, making them somewhat difficult to use for unattended bulk message encryption. In addition most S/MIME implementations originate in the US, making them unusable in other countries.

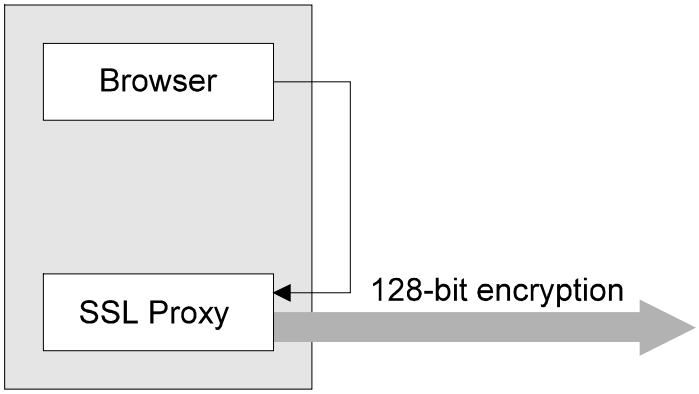
An alternative solution to using encrypted email is to use a second security symbiote-protected link (in fact the software doesn't need to run as a symbiote since there is no need for interactive HTTP-based access) from the server to the merchant or clearing house. This has the advantage of providing a live, interactive link rather than the offline, store-and-forward link provided by using encrypted email. Another good reason for avoiding an email-based solution is the problems which arise with the large numbers of out-of-spec, incompatible, and just plain broken email (SMTP) implementations available — providing your own transport mechanism instantly solves any compatibility problems. Since a substantial proportion of the total amount of credit card fraud is perpetrated by merchants performing fraudulent transactions rather than by customers, the addition of strong security and authentication at the merchant/clearing house level also helps to eliminate this kind of fraud.

SSL Proxies

In late 1996 a number of groups independantly realised that they could solve the problem of weak encryption in browsers by using the proxying capabilities of the browser to route the connection over a secure channel provided by an external proxy. The original reasoning behind the proxying was to allow browsers to connect to remote sites through firewalls by connecting to a proxy (generally a part of the firewall) which would forward the connection past the firewall and on to the remote host. By running the proxy software on the same machine as the browser and adding encryption to the proxy, a secure link could be provided. Instead of the original browser configuration of:



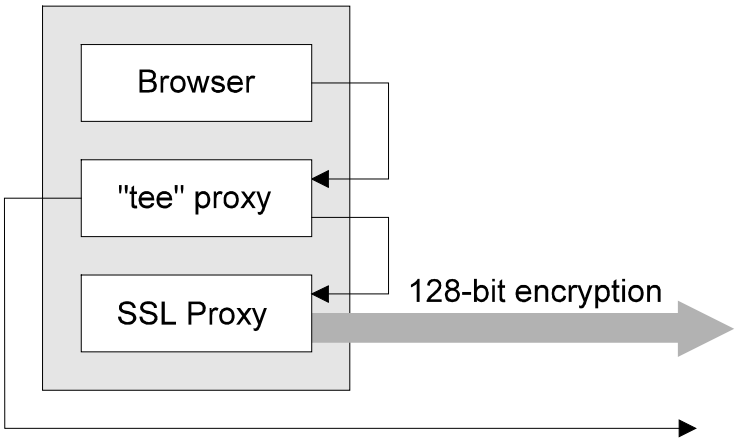
the modified system would redirect the browser to connect through the proxy, which would then provide the strong encryption which the browser was incapable of providing:



The first program to provide this capability was F-Secure `ssh` [DataFellows], which provides a secure link over which arbitrary TCP/IP connections can be redirected. The problem with this configuration was that it required the `ssh` software to be run at both ends of the link and so wouldn't work with standard web servers.

An improved solution which appeared from a number of sources in late 1996 involved the use of SSL proxies which provide 128-bit SSL capabilities and can be used with any server which provides full-strength encryption. All of these proxies are based on SSLeay, which is discussed in more detail on page 42, and all provide more or less the same functionality. When the connection to the remote host is established, the proxy negotiates a secure connection on behalf of the browser, authenticates the client using a client certificate if required, and then transparently secures the link for the remainder of the session. Some of the more advanced proxies even provide security feedback through the browser to make the externally-provided security system almost transparent to the user, or add their own graphical cues about security features which the exportable versions of the browser isn't capable of conveying to the user. In the most extreme case it is even possible to spoof the usual browser security indicators, for example by overlaying the Netscape broken key image with a complete one.

The one possible concern with these proxies is that they replace a link with weak security all the way with one with strong security once it leaves the machine, but potentially no security at all on the local machine, making it open to attack using a malicious ActiveX control of the kind described on page 13. This control would automatically install itself after the user browses the web page it resides on, and would install a "tee" proxy which sits between the browser and SSL proxy and redirects a copy of all data to an external host before it reaches the proxy. Because plugging in the SSL proxy is made very easy by the nature of proxying, plugging in the tee proxy is equally easy, and can be done without the users knowledge even with the highest level of ActiveX security enabled:



There are a number of workarounds to this problem. The SafePassage proxy uses the browsers built-in weak encryption for the connection from the browser to the proxy, decrypts the weakly-encrypted data at the proxy, and then re-encrypts it using its own strong encryption for transmission over the external link, in effect performing a benevolent man-in-the-middle (MITM) attack on the session [SafePassage]. The r³ proxy dynamically varies the proxy address and port number every time the proxy is started, and compares the proxy configuration for the browser with the initial settings when it is shut down. If there are any changes (which would indicate the presence of something like a tee proxy), the user is alerted [r³]. Other examples of SSL proxies are [PersonalSecure] and [SSR].

Common Pitfalls

Although a complete discussion of the pitfalls encountered in implementing security systems is beyond the scope of this paper, this section covers some of the more common problems. There are a considerable number of computer security books available which can provide more information; of these [Kaufman 1995] should probably be required reading for anyone considering designing or implementing a security system for use on the Internet.

Beware of snake oil

"Snake oil" is a term used to describe security systems which are sold on a "trust us, we know what we're doing" basis. If the organisation producing the software is a well-established computer security company with years of experience, then this is probably a fair assumption. Unfortunately, however, the software usually comes from a relatively unknown company, was written by people with little or no cryptographic skills, and provides very little security. "Unbreakable" security systems of this kind have been announced, and then broken, for years. For example a 1987 paper analysed a number of

commercial security packages using proprietary encryption methods: SuperKey, which "protects against any but the most sophisticated attackers", Padlock, "generally accepted by cryptologists as being unbreakable", PS3, which "provides total security in the microcomputer environment", Crypt, and N-Code. All were breakable within minutes on a PC [Kochanski 1987]. The paper concludes with the comment that "security program suppliers should behave more responsibly and realise the importance of supplying genuinely secure systems".

Unfortunately, this hasn't happened. With the recent popularity of Internet-based electronic commerce, a wide variety of snake oil security systems are being marketed by vendors (one popular FTP site even has a directory "sponsored by Snake Oil Productions Inc" for distributing these programs [funet]). While they have appeared in numbers large enough to make it impossible to analyse them all, the ones which have been examined (usually because of especially outrageous claims by vendors) have failed to stand up to close scrutiny. For example in mid-1996 a company announced Cybank, an amazing breakthrough in Internet banking which rejected "insecure public-key encryption algorithms" in favour of their own proprietary system. This was broken in 6 hours of work, which included the time it took to learn the programming language necessary to allow the attack [Collins 1996]. The product had been aimed at providing "secure" Internet banking services.

Another company with an "unbreakable" Internet privacy system offered to sell the company for $1 to anyone who could break it. "There may be rumors that someone has broken the system, but that is not possible, it will never happen" [Wink 1996]. Shortly after the initial announcement, details on how to break the system were posted [Anon 1996b]. It is not known whether the attacker bothered buying the company for $1.

A third company offered anyone who managed to break their software a free copy of the program without, apparently, noticing the irony inherent in the offer.

Snake oil is not limited to Internet-based security systems. In 1994 a German company had a wonderful idea: They would fill a CDROM with encrypted commercial software, and then sell keys to unlock individual programs to people paying by credit card. A number of commercial vendors contributed software, and the CDROM was distributed at the CeBIT computer show. The encryption was quickly broken by members of the Chaos Computer Club ("CCC", who had warned the company in advance that the encryption was insecure), a move which rendered unnecessary the credit card payment portion of the process. The codes were posted to bulletin boards, the Internet, and eventually even the German Telecom's BTX system [Anon 1994a] [Woody 1994a] [Woody 1994b] [ravemaster 1994] [Woody 1994c], and the 300,000 copies of the by now (in)famous Yellow Point CD, each containing approximately DM 100,000 worth of software, quickly became a collectors item. The company which created the disk ceased to exist shortly afterwards [Dittes 1994]. After the Yellow Point debacle, another company offered a similar CDROM, but they knew they were safe because they were using unbreakable encryption for their disk. After two dozen software packages had been decrypted by CCC members, the company admitted defeat.

A third company contacted the CCC directly and offered them a DM 100 reward if they could break the security on their CDROM. The disk was returned with the security broken, and the CCC got their DM 100, along with 1000 CD's and a thankyou note from the company. Finally, after IBM tried the same thing and the CCC again broke the system after 300,000 CDROMs full of commercial software had been distributed, the enthusiasm for this form of software distribution evaporated [Spiegel 1994].

Unfortunately snake oil isn't limited to smaller software vendors. Well-known commercial programs from large vendors also offer very weak encryption (this is genuine snake oil, and not due to US export restrictions). The Windows 95 share passwords stored in the registry were protected by an algorithm so weak they could be broken by someone with no crypto knowledge [Ross 1996]. Various programs exist which will automatically break the encryption offered by software such as Ami Pro, Arc, Arj, Lotus 123, the "improved encryption" in Lotus 123 3.x and 4.x, Lotus Symphony, Microsoft Excel, Microsoft Word, Novell Netware, Paradox, Pkzip 1.x, the "improved encryption" in Pkzip 2.x, Quattro Pro, Wordperfect 5.x and earlier, the "improved" encryption in Wordperfect 6.x, and many others. A number of programs (too many to list here) which will break the encryption of a variety of software packages are freely available via the Internet. These programs include, for example, a WordPerfect encryption breaker [WordPerfect] and a Word for Windows encryption breaker [W4W]. A general collection of encryption-breaking programs is available from [hacktic]. There are also commercial companies which sell software to break these encryption systems [AccessData] [CRAK Software]. These systems are often so simple to break that at least one package which does so adds several delay

loops just to make it look as if there were actually some work involved in the process. Although the manuals for these programs make elaborate claims about the security of the encryption, it can often be broken with such time-honoured tools as a piece of paper, a pencil, and a small amount of thought.

One claim often made about snake oil systems is that "the algorithm is proprietary, so noone can break it". As the reverse-engineering of RC2 and RC4, two notable (ex-)proprietary algorithms showed, this is no guarantee of security. Further algorithms which have been leaked, with far more serious consequences than for RC2 and RC4, are A5, used for GSM cellphone encryption, and CAVE, used for US digital cellphone encryption. Originally it was claimed that the details of A5 could not be disclosed because it was a strong algorithm and details might end up with terrorists or with the likes of Saddam Hussein. Then in June 1994 the story was changed and A5 became too weak to disclose, so that revealing details would allow it to be easily broken. This proved to be the case when A5 was leaked shortly afterwards. Details on what it would take to break A5 are given on page 17. At about the same time, A5 was further weakened to create a variant called A5X or A5-2 which is used in newer GSM phones and is even easier to break.

In contrast the US digital cellphone encryption system was designed right from the start to be extremely easy to break. Although out-of-band control information is secured with a simple algorithm called the Cellular Message Encryption Algorithm (CMEA), the actual data payload is "encrypted" with a different algorithm, the Cellular Authentication and Voice Encryption (CAVE) algorithm, which generates a pair of 260-bit voice privacy masks (VPMs) and applies them repeatedly to each frame in either the forward or reverse channel by exclusive-oring them with the data. These VPMs can be recovered with almost no effort, and may even be zero during the initial portions of the call [TR 45.3 1992]. This system was designed by the TR45.3 Ad Hoc Authentication Task Force to make it "pitifully easy to break" because they feared that creating an even slightly secure system would result in any products using it being embargoed for export [Barlow 1992]. Both A5 and CAVE are examples of officially sanctioned snake oil, a process akin to building a bridge which is designed to collapse if a certain weight is placed on it.

The tools to reverse-engineer encryption systems are generally available to anyone who requires them, often as part of existing software such as programming tools, or through research institutions. These tools range from simple debuggers through to sophisticated tools such as decompilers which can reconstruct a reasonably good approximation of the original source code from an executable program [dcc], and hardware analysis tools such as electron beam testers (which are particularly popular with European smart card hackers) and focused ion beam workstations.

Although RC2 and RC4 were designed by an expert cryptographer and are secure, many proprietary algorithms are designed by amateurs, and are not. Even if the system isn't reverse-engineered, once it goes to court lawyers can request full details of the system design, circuitry, program source code, and other information for examination by (hostile) expert witnesses. Court cases in the UK have shown that if the security system can't withstand this combination of legal and technical scrutiny, the case may collapse. Even Ministry of Defence contractors with security clearances weren't immune to court-ordered searches [Anderson 1993].

Vendors have also occasionally issued challenges with accompanying financial rewards in order to "prove" the security of their products. Unfortunately this demonstrates very little about the security of a product, since an ideal strategy for someone who can break the security of the system is to ignore the modest reward offered by the vendor and wait a year or so until the system is in active use. The rewards obtainable at that point will be far greater than anything the vendor offered in the challenge. In addition, the vendor will typically skew the conditions of the challenge, rendering the whole exercise invalid in terms of testing the products' security (one firewall-hacking challenge in 1995 seemed to consist of the vendor plugging a cinder block into the Internet and inviting attackers to try to break into it). Careful system design, in-depth testing, and independent review will demonstrate the quality of a product far better than a public challenge to attack it.

Since there are a large variety of well-known, heavily analysed, and trusted security systems around, there is no reason to rely on snake oil to protect your electronic communications. Systems which rely on "proprietary", "unbreakable" encryption algorithms and protocols should be avoided in favour of ones which use established, proven security methods. A checklist of warning signs and points to watch out for is given in [Curtin 1996].

Security systems are subject to bit rot

"Bit rot" is a mythical affliction which affects software, causing it to cease functioning over time even if nothing is changed [Raymond 1991]. Security systems suffer from a similar slow decay in their effectiveness as new attacks and flaws in the system are discovered so that even if no changes are ever made to a system, its effectiveness will gradually decrease over time. One cause of this is an "if it ain't (obviously) broken, don't fix it" attitude which causes organisations to continue using systems with well-known security holes long after improved versions have become available. An example of this is the more than 10,000 sites which continue to run version 1.3 or earlier of the NCSA web server, which has been known for some time to have serious security flaws. Warnings about the security problems were widely distributed [CIAC 1995] along with scripts to exploit holes [Lopatic 1995], and yet large numbers of `.edu`, `.com`, `.gov`, and even `.mil` sites continued using the flawed versions [Prettejohn 1996]. One survey of 2200 systems performed in late 1996 found that *two thirds* of these had serious security problems, and that more than one third of all bank sites had security problems. In addition, systems such as those run by banks, government departments, and similar organisations, which are likely to be targets for attacks, had twice as many potential security vulnerabilities as a selection of randomly-selected systems [Farmer 1996]. Another example is the number of organisations which continue to use last weeks version of `sendmail` even though security holes in it have been widely publicised. Security is an ongoing issue with no easy "fire-and-forget" solutions.

One way of keeping up with security problems is by tracking security alerts from organisations like the Computer Emergency Response Team (CERT) [CERT], the Forum of Incident Response and Security Teams [FIRST], and the Computer Incident Advisory Capability [CIAC]. There are a number of related mailing lists, as well as other groups covering non-Unix systems. Up until a few years ago there was a form of gentleman's agreement between system administrators and software vendors in which administrators would keep quiet about security bugs until the problem had been fixed by vendors. Unfortunately many vendors would use this silence to ignore the problem, so that only extremely well-known problems were ever fixed. Because of this it has in recent years become common practice to publish code to exploit any newly-discovered security holes, which has lead to a remarkable improvement in vendor responsiveness. Security alerts frequently carry the dire comment "code to exploit this vulnerability has been published", which translates roughly to "this needs to be fixed yesterday".

Bit rot affects not only software but also the legal status of security measures. New laws covering issues like the validity of digital signatures may be passed, or existing laws challenged in court. For this reason security software may need to be updated not only to adapt to new types of attacks, but also to accommodate a changing legal environment.

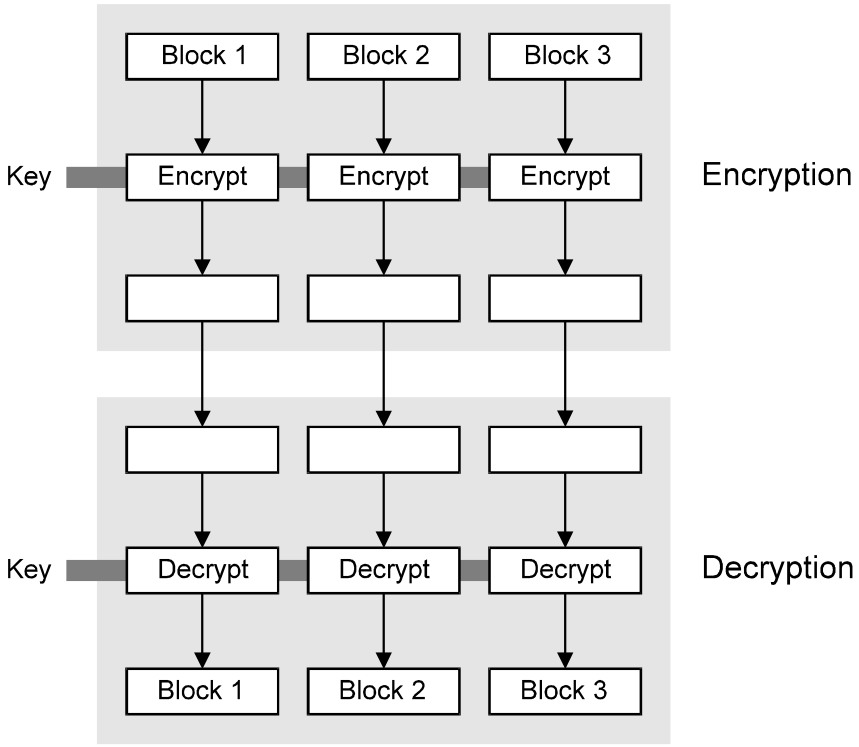
Not all algorithms and protocols can be used with impunity

A point to take into account when choosing encryption algorithms and protocols are patent issues. All public-key algorithms and a number of useful protocols are patented in the US, but are generally not patented outside the US. This has lead to a number of implementors outside the US choosing the most convenient algorithm or protocol in the hope that any infringement or licensing issues in the US will sort themselves out in the future. Unfortunately these issues usually don't sort themselves out, and place attempts to create a product which is viable both inside and outside the US at a considerable disadvantage if licenses can't be obtained or the product is found to infringe on patents once it has been deployed — electronic commerce is a global affair, which means it isn't possible to force it to fit a local business model without creating future interoperability problems. It is therefore prudent to consider possible future legal issues when choosing algorithms and protocols during the design stages of a project. Well-known, trusted conventional encryption algorithms like triple DES are usable without any restrictions; there seems little reason to choose a patented or proprietary alternative. By sticking with well-established, non-restricted protocols for operations like key exchange and certification, it is possible to avoid any future problems with licensing and patent infringement. The only area which will cause problems is with public-key algorithms and with a number of digital cash-related protocols, however the recent split between RSADSI and Cylink has now provided a choice of vendors from which to license public-key technology (although the patent licensing costs still put them out of reach of all but large corporations). In practice, with a little care in the selection of algorithms, only the public-key portion of a security system will require special licensing considerations.

Simple compliance with security standards is no substitute for independent analysis

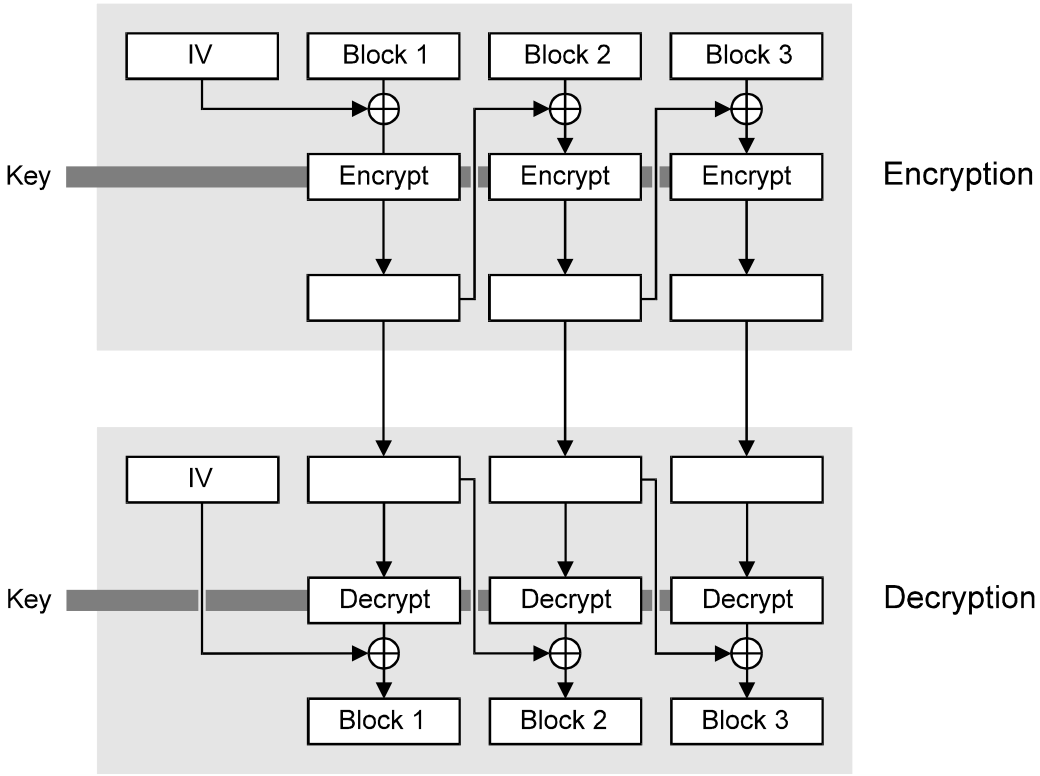
Implementing a certain system "because it's the standard" isn't always a guarantee of security. Problems can arise either due to misinterpretations of standards or incorrect implementations of systems based on standards, because the standards themselves contain flaws, or because the standards being applied are inappropriate. The most common example of the first type of error is the number of DES implementations which are either incorrect, or which use the algorithm in a weak encryption mode such as ECB (DES probably carries the distinction of being the worlds most commonly misimplemented security system). A programmer without security experience who is told to implement DES for a project will typically work on the code until they have something which transforms data into noise and back, and will then employ it in the most obvious encryption mode, ECB.

ECB, or Electronic Codebook mode, breaks a data stream into separate subblocks which correspond to the unit of data which DES can encrypt or decrypt in one operation, and encrypts each subblock separately. The ECB encryption process is:



Note that each block is independent of all other blocks, so that changing any of the encrypted blocks will not affect any other block. There are several forms of attack which can be used when an encrypted message consists of a large number of completely independent data blocks. For example it is often possible to identify by inspection repeated blocks of data, which may correspond to patterns like long strings of spaces in text. This can be used as the basis for a known-plaintext attack.

There also exist other modes such as Cipher Block Chaining (CBC) in which one block of encrypted data is chained to the next (such that the ciphertext block *n* depends not only on the corresponding plaintext but also on all preceding ciphertext blocks 0…*n*-1), and Ciphertext Feedback (CFB), which is a means of converting a block cipher to a stream cipher with similar chaining properties. The CBC process is:



In this mode each block is affected by the previous block, and the first block is affected by a random value called the initialisation vector or IV. By making the IV different for each encrypted message, the encrypted message contents will differ as well because the initial random IV propagates throughout each message. To see why this is important, consider the following message-modification attack on a message encrypted in ECB mode, which doesn't require any knowledge of the encryption key being used.

Lets assume that Bob asks his bank to deposit $10,000 in account number 12-3456-789012-3. The bank encrypts the message `Deposit $10,000 in account number 12-3456-789012-3` and sends it to its central office. The message to be encrypted in ECB mode looks as follows (with block boundaries indicated by shading):

```
Deposit $10,000 in acct. number 12-3456-789012-3
```

Bob intercepts this message, and records it. The encrypted message looks as follows:

```
H2nx/GHE KgvldSbq GQHbrUfo tYFtUk6g S4CpMIuH 7e2MPZCe
```

Later on in the day, he intercepts the following message:

```
H2nx/GHE KgvldSbq GQHbrUfo tYFtUk6g Pts2LtOH a8oaNWpj
```

Since each block of text is completely independent of any surrounding block, he can simply insert the blocks corresponding to his account number:

```
........ ........ ........ ........ S4CpMIuH 7e2MPZCe
```

in place of the existing blocks, and thus alter the encrypted data without any knowledge of the encryption key used. Bob has since gone on to early retirement in his new Hawaiian villa.

Now assume the message is encrypted in CBC mode. Since a different random IV is used for each message, Bob can't tell that the second message contains a transfer order since the entire message differs from the first message. Even if he can somehow guess the nature of the message, he still can't substitute his account number because the chaining from one block to another will result in the first inserted block decrypting to garbage.

Whether the transformation being applied to the data is DES is another question (one commercial encryption-breaking company even uses software-configurable hardware to cope with the number of

incorrect DES implementations they encounter). The DES algorithm has the property that any small changes to it significantly weaken the strength of the encryption, so that any implementation which doesn't follow the standard exactly provides greatly reduced security. In addition although the use of the weak ECB encryption mode is specifically warned against in a number of standards [FIPS 1980] [ISO 1987] [ISO 1991a] [ISO 1991b], this mode is the easiest to implement and is the one most often employed by implementors with little cryptographic knowledge. For example one NZ government department uses a combination of an incorrect DES implementation and ECB encryption to protect their data. Another department uses DES with a 6-digit PIN to attempt to provide message authentication and non-repudiation as per their interpretation of an EDI standard. This system will provide the security expected of it only until the first time it is subject to an attack. It is probably safe to assume that a considerable number of DES implementations use keys composed of ASCII characters, making it a good idea to base key searches on the data patterns present in ASCII text, and to try out key characters in a manner which is based on standard text statistics.

Another organisation protects medical records using an 8-digit numeric key for DES (giving just under 27 bits of effective key size), unaware that the last bit of each key byte is used as a parity bit, for a total of just under 19 bits of effective key. Using the software described in the previous section would allow a PC to perform a key recovery attack and decrypt the medical records in under a second, in which time a user probably wouldn't even notice that an attack had taken place. The misuse of DES keys due to the parity bits seems to be quite common, and can take forms more dangerous than simply weakening the existing key. For example a few years ago a Kerberos-based DES implementation which checked the validity of the parity bits was used in an encrypted `telnet` implementation which obtained the DES keys via a Diffie-Hellman key exchange. Since the resulting key bits which were fed to the DES implementation were random, only 1 in 256 keys had the correct parity. The rest were silently dropped, so that $^{255}/_{256}$ telnet sessions used an all-zero key [Blaze 1996b]. A closer examination of some of the flaws in implementing cryptosystems which an independent review should catch is given in [Anderson 1994c]. Principles for ensuring proper use of public-key cryptosystems are given in [Anderson 1995].

The problem of using programmers with few or no cryptographic or security skills to implement security software extends further than incorrect implementations of security standards. The fact that a programmer may be skilled at Windows/Unix, C++, or network programming does not make them an automatic candidate for implementing security software. One recent job advert provided a long list of necessary skills covering every imaginable programming environment, language, and methodology, and then finished the list with "crypto experience a plus" [Consultant 1996]. This is not a good basis for implementing security software. For this reason the three potential solutions recommended on page 33 are complete encryption libraries for which the security design and analysis has been carried out by crypto-knowledgeable programmers and which have been subject to fairly intensive independent review.

Examples of the second type of error, the use of flawed standards, are less common. One notable example is the X.509 protocol for signed, secure communication between two systems [CCITT 1988]. This protocol contains two weaknesses, one of which has, unfortunately, made it into other standards such as [ISO 1994b] which use X.509 as a guideline. The problem lies in the fact that X.509 signs data after encrypting it, making a variety of attacks possible [BAN 1989]. These attacks can be defeated by the simple measure of signing the data before it is encrypted, as is done by a number of non-ISO protocols and de facto standards. Another example of a flawed standard is [ISO 1994a], which allows easy forging of messages and has a generally unsound architecture [Rueppel 1993]. Even the best can make mistakes when designing cryptographic protocols [Blaze 1994].

An example of the third type of error is the popular Orange Book [TCSEC 1985] C2 security marketing standard, which was originally designed to be applied to non-networked multiuser mainframes with users allocated to individual terminal connections which could be assigned a single security level, but is now being applied to networked single-user systems using multiple windows (which may require different security levels), accessing and sharing files and data over network connections, and downloading and running their own code (at least one common network server operating system loses its security certification the moment a network interface is plugged in. The network is exactly the location from which an attack is most likely to originate). Worse, the extensive evaluation process (often a year or more) means that the operating system version which is certified is often several generations of updates and bug fixes behind the one everyone is using, making the certification irrelevant for everyday use [Rothke 1996]. The TCSEC was driven by the particular needs of the US Department of Defence, which focused on threats as perceived by the DoD and was based on the DoD

concept of operations which included secure physical environments, cleared personnel, a rigorous classification system, and use of centralised, non-networked mainframes. Unfortunately the implicit contract between manufacturers and the government (that if manufacturers built equipment to the TCSEC, the government would buy it), wasn't upheld by the government, leading to little market acceptance of the security standards. A prime example of the problems inherent in creating TCSEC equipment was the VAX A1 security kernel, which took 9 years to go from preliminary design to the evaluation stage, at which point it was terminated due to corporate lack of interest.

Because the Orange Book allowed the creation of highly secure multiuser timeshared mainframe systems but not (unless increasingly tortuous interpretations were applied) what is currently expected of a user-friendly, usable system, a European effort which combined earlier work by Dutch, French, German, and UK government groups produced the ITSEC criteria[11] [ITSEC 1992] with a new important requirement: ease of use. Unfortunately these criteria still don't provide much useful detail for Internet security systems: issues such as computer hardware are "beyond the scope of this document"; encryption algorithms are "the subject of certification by the appropriate national body".

For these reasons it is imperative that any security system be subject to independent review and analysis of both the protocols and the implementation (a process which has been described to as "formalised paranoia" [Simmons 1994]). Independent analysis of code is also essential to catch various programming and implementation flaws, both accidental and deliberate. For example the problems programmers have in generating secure random numbers have received so much publicity (at least in the computer security community) that the random-number generation routines in an implementation are an immediate target for third-party analysis (both benevolent and malicious). Any flaws will generally be found fairly quickly *provided the implementation is available for third-party analysis*. The reason why so many people trust security systems such as PGP is that anyone can examine the code to reassure themselves that it is indeed a correct implementation. The fact that the software has been subject to intense, independent review makes it much stronger than any proprietary, untested implementation.

Another reason for an independent review of code is to catch various traps which may be deliberately hidden in the code by an unscrupulous implementor. For example the NIST/NSA Digital Signature Algorithm (DSA) [NIST 1994] provides a particularly hospitable setting for subliminal channels, which allow data such as the DSA private key or other secret information to be leaked [Simmons 1993a] [Simmons 1993b]. Subliminal channels can also be embedded in RSA, and the authors of one paper which covers this topic have created a modified copy of PGP which contains a subliminal channel which leaks PGP private keys [Young 1996a]. Subliminal channels can also be embedded in the standard PKCS message padding used in a large variety of public-key based encryption software [Gutmann 1996b] and in a number of other places. These subliminal channels are often very easy to insert[12] and are completely undetectable unless the source code can be examined. Other deliberate but hard-to-detect weaknesses can also be used to weaken encryption software, such as generating RSA keys in a predictable (to the software author) fashion [Piper 1993].

A final reason for independent review of an implementation can be taken from the discussion on page 15: If a security system is expected to withstand hostile analysis in a court case, it should be subject to friendly analysis before it is deployed.

Users will misuse a system they don't understand

One of the largest barriers to the adoption of secure electronic commerce systems is the lack of user understanding of security issues, and the lack of developer understanding of the lack of user understanding. For example users will generally say that the security of a system is a top priority, but will be unwilling to pay extra for the security or to invest the effort necessary to manage a secure system. If a company produces security software, users will employ it in a manner which suits them rather than the way it was intended to be employed. For example a number of organisations who are currently replacing traditional X.25-based systems for electronic purchasing and inventory management with Internet-based ones are generating public and private keys for all users at a central site and then couriering or mailing them out to users on disk, instead of having the users generate the keys and having them certified by a certification authority (CA) as intended by the software vendors. This was because the central sites needed to show due diligence in their management of the keys, and their lawyers

[11] There were some difficulties involved in this effort. Some criteria were unavailable for examination, or appeared to contain only watered-down versions of the real, classified criteria.

[12] There exists at least one encryption library with "Insert subliminal channel here" comments in the code.

advised them that while the legal system would recognise transmission of private keys via trusted courier or registered letter, they were unlikely to look favourably on a scheme involving key certificates and CA's, which could see any use of its certificates turn into a messy and expensive test case of uncertain outcome if the system were ever challenged in court. This centralised key generation process completely defeats the purpose of secure client-side key generation, CA's, and a key certification hierarchy, but was necessary for legal reasons. Another reason for using centrally-generated keys was that the central site could then charge users for each key generated, whereas it was seen as highly unlikely that users (who understood even less of the issues involved than the people running the central site) would pay for key certificates.

Although current Internet users who can reconfigure their network connections and install their own software and will download every new pre-release version of the latest web browser to play around with can cope with the intricacies of public key certificate management, the non-computer-literate masses who will eventually comprise the majority of a security systems' users won't be able to handle this. In addition, businesses are increasingly moving towards using the low-cost, easily-available nature of the Internet as a general-purpose WAN for electronic commerce applications where an X.25 solution was seen as too expensive to install and maintain. Whereas a traditional system used in a retail application might have had many PC's on a LAN communicating with a central site via an X.25 gateway, an equivalent Internet system with one server acting as a gateway is generally seen as being non-cost-effective both for large retail chains who may have many thousands of sites which would all require the installation of extra servers to control Internet access, and for smaller businesses who don't want to install yet another PC. For this reason each machine is given its own net connection, either through a modem or (more recently) through an ISDN link. These decisions are driven entirely by financial concerns: installing large quantities of routers, servers, and fixed data links is a lot more expensive than buying a shipment of modems (the configuration and maintenance costs of the modem-based solution are rarely considered).

So far the majority of problems with electronic commerce applications are due to network problems and user misunderstanding [Borenstein 1996]. It is reasonable to expect that the majority of security problems will also be due to user misunderstandings, of which some, such as users blindly clicking "OK" for any security warnings which appear, have already been mentioned above. Low-tech attacks which exploit design or operational errors are already more common than sophisticated technical attacks [Anderson 1994b], and will only become more so as the Internet user base migrates towards non-technical users. Software developers will in the future need to go to more trouble talking with customers before designing security products and protocols, and be more creative in finding ways to adapt their systems to meet customer requirements (in several instances of the previous example of centrally-generated keys, the two largest players in the field lost out because their software required that public/private keys be generated by the client, and couldn't be made to work with a third-party key management system because the key storage formats were undocumented).

Suggested Solutions

There are a number of possible solutions to providing secure electronic commerce services over the Internet. The best-known one is probably the Secure Electronic Transaction (SET) protocol, which represents a clever compromise between the US governments requirement for easily-breakable security and the need to maintain the confidentiality of financial information. SET is also an example of a security protocol done right: The design uses standard, well-known algorithms, and all design details were published and widely distributed for comment. The system then went through several revisions based on public feedback before the design was finalised, making it the first significant banking protocol to be implemented in this manner. The SET protocol is a politically astute design which has been created to handle only one thing: authorisation of electronic payments between a client, a vendor, and a clearing house. Other details such as communicating information on the goods to be purchased or the amount of the purchase are excluded from the SET design. By moving sensitive information into the section of the public-key encrypted data block which is normally reserved for random padding (the so-called "extra strong" or *EX encryption*), SET protects this information using the stronger 768-bit or 1024-bit public-key encryption rather than the weaker DES encryption. In addition the limited amount of space available in the public-key encrypted block means this mechanism can't be "misused" for general-purpose encryption, thereby satisfying US government requirements.

SET uses a session-based communications model and an X.509-based certification model, but overcomes many of the problems inherent in X.509 by relying on credit card vendors to act as CA's (that is, to testify to the validity of a credit card or merchant). This security model, based on credit card usage, is far easier to fit into the X.509 framework than a more generalised CA mechanism, making the SET equivalent of a CA far easier to operate than a normal CA. The SET protocol also provides basic transaction-processing style functionality, something not present in more basic session-based security systems such as SSL. Unfortunately this enhanced capability has also made SET very complicated to implement, which has significantly delayed its acceptance in the marketplace and will probably lead to future teething problems due to interoperability conflicts among different implementations.

Although SET fulfills its designed requirements perfectly (and was meant to be exportable from the US, however the US government recently blocked the export of the SET reference code), it can't be used as part of a general-purpose secure electronic commerce system. The necessary building blocks for a number of possible alternatives which will provide general-purpose encryption capabilities which can be adapted to the task at hand are openly available outside the US, and although full coverage is beyond the scope of this paper, three main examples will be considered here.

These three examples are based on special selection criteria. A common problem when investing in security technology is that in order to sell security, the risk of not buying it must outweigh the cost of the security system. A back-of-the-envelope calculation at a recent security conference showed that the average (large) company spends $50,000 per year on security, the average incident costs them $100,000, and the average security product costs $200,000. Although companies aren't paying as much as their losses due to security breaches, they would lose even more if they paid for proper security products [Ranum 1996]. This makes security products somewhat difficult to sell to management. A solution to the problem is to use one of a number of free, high-quality, proven code libraries to provide security.

The three examples have been chosen because they are freely available[13] for anyone to use and have been subject to extensive review and analysis because they are provided in source code form. Since they are in widespread use, incompatibility problems across different systems have been resolved, with versions being available for most of the systems in common use on the Internet. In addition the implementations have easy-to-use high-level interfaces which don't require the user to have extensive cryptographic knowledge. This is an important issue because many organisations, when asked to provide security, will use whatever systems have been recommended by the appropriate authorities and then employ them in ways which leave large holes which attackers can exploit. By providing a clean, easy to use high-level interface, the developers of the three implementations described below make it easy for non-cryto-aware programmers to build strong security measures into their software without needing to spend months or even years learning the details of cryptographic and security algorithm and protocol design and implementation.

PGP

One solution which is reasonably popular due to the ease of implementation is PGP [Garfinkel 1994] [PGP] [Stallings 1994] [Zimmermann 1995a] [Zimmermann 1995b]. PGP is the de facto standard for email encryption on the Internet, is available on every platform of note, uses proven algorithms (without any facility for weak, US-government-approved security), and is already used by a number of companies to handle emailed credit card numbers [Wayner 1993]. Although this is hardly online electronic commerce, it does provide an internationally viable way to securely transmit credit card orders.

PGP is packaged into a single unwieldy bundle which performs encryption, decryption, signing, verification, and key management, and can be bootstrapped into any environment without too much trouble since it doesn't depend on any existing infrastructure. Unfortunately version 2.x of PGP was intended to be used as a standalone, command-line program by knowledgeable users, which makes it very difficult to adapt as a general-purpose security solution. A program which relies on PGP to provide security typically provides a graphical interface which quietly feeds its data out to the command-line-driven PGP program which processes it in the background and then returns it to the user [Luckhardt 1996]. Apart from being a somewhat painful way to encrypt and decrypt messages, it can also lead to security problems due to the way parameters are passed on the command line by the shell

[13] In at least one case the free nature of the software has caused a division by zero error in an accountant assigned to approve its use.

program. For example if a mail program calls PGP with the name of a message to encrypt and an address from an address book:

```
pgp -ea message John Doe <jdoe@somewhere.com>
```

then PGP will encrypt the message for the first key it finds with `John` in the user name, the first key it finds with `Doe` in the user name, and finally the key with `jdoe@somewhere.com` in the user name, because the spaces in the address are treated as command separators unless the mail program is intelligent enough to put the entire address in quotes (many programs don't do this). There are a number of other problems of this sort which arise due to the fact that the only interface to PGP is provided through a command line.

Since PGP doesn't support single-pass processing, it must write data to disk as it runs, making in-place manipulation of data impossible. This makes it unusable in a transaction-oriented online system, since the delay inherent in writing a message to a file, invoking PGP to process it, and reading it back into memory for transmission is intolerable.

The nature of the PGP code, which consists of a dense tangle of functions which often perform nearly the same task as a slightly different function in another module, all tied together with global variables and complex content coupling, make it very difficult to separate the required functionality from the main program. Implementation of an integrated public-key directory is difficult, requiring the use of two address books for encrypted electronic mail, one to select email addresses (handled by the mail program) and one to map email addresses to keys (handled by PGP), which can lead to the previously mentioned problem with addresses. In addition, PGP's key management process is hard to learn and requires a great deal of user intervention. When it comes to providing general-purpose electronic commerce security, PGP 2.x is a minefield disguised as a rose garden.

The impending arrival of PGP 3.x should fix a number of these problems. The PGP 3.x development team threw out the 2.x code and wrote 3.x from scratch. The code is modular, well-designed, and has a simple, well-documented interface. Implementations will be available both as a standalone application which mirrors the current PGP 2.x program, and as a library for integration into other software. PGP 3.x also breaks apart the message processing, key management, and user interface functions into three distinct parts which don't depend much on each other. There are a few standard interfaces between the various parts, but the heavy interdependency of PGP 2.x modules has been removed, making it very easy to provide independent command-line or GUI interfaces, separate key-management functionality, and other features which were impossible in PGP 2.x [Atkins 1996a] [Atkins 1996b].

The PGP 3.0 application programmer interface (API) works by allowing the user to construct a pipeline of modules which process messages. Data is pushed into one end of the pipeline, each module performs a particular operation such as encryption or signature checking on the data as it passes through, and the resulting data drops out of the other end of the pipeline. Creating a particular type of pipeline involves plugging various modules together in the order in which they are to be applied, pushing the data to be processed through the resulting pipeline, and then unplugging the modules again once the data has been processed. For example a pipeline which accepts input text, converts it to a format independent of the host system (so that, for example, a file sent from an MSDOS system is readable on a Macintosh), signs it, compresses it, and ASCII-encodes it for transmission as email (the equivalent of `pgp -sa` when using the command-line PGP program) would be:

Pipeline head → Text canonicalisation → Signature creation → Compression → ASCII-encode data → Pipeline tail

Since this is a fairly standardised task, PGP 3.x provides an alternative single function `pgpEncryptPipelineCreate()` to perform the task.

Once this pipeline has been constructed, data is written into it using a standard `write()` call, the series of operations specified in the pipeline is performed, and the transformed data is removed from the other end of the pipeline via a callback function. This code can be plugged into any application to allow it to generate and process PGP-compatible messages.

There are a few disadvantages to this approach, the main one being that PGP 3.x still relies heavily on PGP 2.x data formats and functionality, with a number of the same constraints on efficiency and simplicity that PGP 2.x had. PGP 2.x used a single public-key algorithm, RSA, which is patented and very difficult to license in the US, a single conventional encryption algorithm, IDEA, which is patented

in many countries, and a single hash algorithm MD5, which was recently broken (to be precise, collisions in the compression function were demonstrated, and it is likely that the useful lifetime of MD5 is rather limited). PGP 3.x extends the range of algorithms to include triple DES in place of IDEA and SHA in place of MD5, both of which are unhampered by patents or (known) weaknesses. Although this will cause backwards compatibility problems with existing PGP implementations, the anticipated immediate widespread acceptance of PGP 3.x will mean that most users will have switched over to 3.x soon after its release. The wide installed user base and large variety of third-party support tools for PGP makes the PGP 3.x library an attractive choice for applications which will work within the existing PGP 2.x framework.

SSLeay

SSLeay is a free SSL implementation written in Australia [Reif 1996] [SSLeay]. It provides full SSL functionality and includes a large amount of support software including sample applications and tools to handle certificate generation and management. SSLeay has been integrated into secure telnet and FTP servers and clients, and a number of web servers such as the NCSA `httpd` and the Apache [Apache], Sioux [Sioux], and StrongHold (originally known as Apache-SSL-US) [StrongHold] and International Stronghold [International Stronghold] web servers (the Apache server was originally based on the NCSA `httpd`, but has since evolved into a system which rivals most other Unix-based servers in terms of functionality, efficiency, and speed. Sioux and StrongHold are commercial versions of Apache. International StrongHold is a version of StrongHold implemented in the UK because the US distributors "did not want to give our customers a false sense of security by shipping a defective [weak-encryption] product"). Apache currently holds around 40% (and growing) of the total web server market share [Netcraft], and is seen as the number one threat by both Microsoft [Clark 1996b] and Netscape [Booker 1996] [Wingfield 1996a]. In fact Apache presents something of a problem for Microsoft and Netscape, whose business model relied on giving away browsers to create demand for their servers and then selling the servers to businesses. The appearance of a free commerce server which outperforms most commercial ones has created something of a problem for software vendors.

SSLeay supports a wide choice of algorithms including the conventional algorithms DES and triple DES (the code for which are used in the commercial BSAFE library sold in the US), IDEA, RC2, and RC4, public-key algorithms Diffie-Hellman, RSA, and DSA, and hash algorithms MD2, MD5, and SHA. SSLeay is available in 16- and 32-bit Windows, and Unix and VMS versions, can be used for non-commercial purposes inside the US if linked with the RSAREF library rather than the built-in RSA code, and can be used for commercial and non-commercial purposes elsewhere. Although intended mainly to provide SSL support for existing applications, it includes a large collection of sample applications which can be adapted as needed for specialised tasks and is built on top of encryption libraries which can be used standalone if required.

The SSLeay API hides most of the low-level details present in SSL (although parameters such as the encryption type, connection type, and session cacheing, can be adjusted as required). Establishing an SSL connection involves associating a network socket with an SSL information structure and connecting to the appropriate service:

```
SSL_set_fd( ssl_connection, network_socket );
SSL_connect( ssl_connection );
```

after which data can be read and written through the SSL connection:

```
SSL_read( ssl_connection )/SSL_write( ssl_connection );
```

(some details involved in establishing the initial SSL connection have been simplified in the above example). The full SSL protocol is nicely encapsulated by this API, and programmers using it require only a minimal awareness of the mechanics of SSL — once the `SSL_connect()` action has been performed all data is transmitted using the SSLeay equivalents of the traditional socket `read()`/`write()` functions. The interface can also be adapted to use I/O methods other than sockets if required.

The SSLeay package includes the necessary tools to allow the user to act as their own certification authority (CA) to generate and certify keys which can then be used with SSL-enabled browsers like Netscape and MSIE. In fact this key certification capability extends beyond SSL to providing powerful and general-purpose certificate management systems capable of working with X.509 key certificates, which look set to become the standard method for exchanging and storing public keys. This means that

an organisation can use SSLeay to implement their own key management system within the organisation or for users of their software without having to go to external certification authorities, allowing organisations greater control over keys and reducing the expense of using external CA's. Although obtaining key certificates from a commercial CA is difficult for free servers like Apache, the Verisign CA in the US [Verisign] will issue certificates for software whose quality they are happy with such as the Sioux server (sold from South Africa). Thawte Consulting, who sell the Sioux server, will also issue certificates [Thawte]. Other non-US commercial CA's are EuroSign [EuroSign] and COST [COST].

Ideally, SSLeay should be used in combination with a web server. The SSLeay implementation of SSL is significantly better than the Netscape reference implementation, and when combined with a server such as Apache it provides a secure, high-performance platform unhampered by US export restrictions. Unfortunately the browsers which connect to this server still originate within the US, which means that even though the server supports strong encryption and uses a sound SSL implementation, the browser doesn't. This makes it necessary to employ a symbiotic application which sits on top of a client-side SSLeay implementation to talk to the server, effectively bypassing the browser to provide proper security functionality.

[Editorial note: SSLeay is currently undergoing extensions of the message encryption functionality; capabilities may have changed by the time this paper is published]

cryptlib

cryptlib is a free cryptographic toolkit whose goal is to allow the easy integration of cryptographic services into applications with a minimum amount of effort [cryptlib]. Like SSLeay, cryptlib supports a wide choice of algorithms including the conventional algorithms DES and triple DES (using the same code as SSLeay), IDEA, RC2, RC4, RC5, Safer, Safer-SK, and Blowfish, public-key algorithms Diffie-Hellman, DSA, and RSA, and hash algorithms MD2, MD4, MD5, SHA, and RIPEMD-160. cryptlib is available as 16- and 32-bit Windows dynamic link libraries (DLL's) and Macintosh shared libraries as well as in source code form for the Amiga, BeOS, DOS, Macintosh, OS/2, Windows, and Unix, can be used for non-commercial purposes inside the US if only the Diffie-Hellman and DSA, but not RSA public-key algorithms are used, and can be used for commercial and non-commercial purposes elsewhere.

Unlike the PGP 3.x library and SSLeay which support direct implementation of existing applications (PGP and SSL respectively), cryptlib is a toolkit which allows users to implement whatever security system and protocol they require. This means that using cryptlib requires slightly more cryptographic knowledge than PGP 3.x or SSLeay, although relatively simple functions such as a PGP-like encryption/digital signature capability can be implemented in about a dozen function calls. cryptlib provides both a low-level interface which allows direct access to the raw encryption algorithms, and a (preferred) high-level interface which uses cryptlib's object-management routines to take care of the details of encrypted data management. When used in this manner all data objects are treated as opaque "blobs" which are meaningful only to cryptlib functions which automatically know how to process a given blob. For example the `cryptCreateSignature()` function would create a signature blob whose nature can be queried with `cryptQueryObject()` and which can be processed with `cryptCheckSignature()`.

cryptlib hedges its bets on key management by supporting both PGP and X.509/SET keys with the option of storing them in one of a variety of commercial-grade relational databases such as Oracle, SQL Server, or MS Access, all accessible through a standard interface. This has the advantage that it can be tied into any application which uses an existing database to manage some form of directory such as an email address list, eliminating the type of problem which PGP has with dual directories for email addresses and keys. The decision to use a proper RDBMS to store keys instead of hoping for the appearance of an X.500 directory implementation is borne out by one large corporations' attempt to go with X.500 — after a considerable amount of effort they declared the X.500 solution to be unworkable and switched to a DBMS for their key management needs [Coe 1995]. Since cryptlib was designed to work well with Windows it can be easily integrated into general Windows software built with tools such as Visual C++, Visual Basic, and Delphi. Unfortunately the fact that PGP and X.509/SET certificates are handled only as foreign key formats means that the library can't currently perform the certificate checking necessary to fully verify public keys, but instead relies on the software which originally created the key for this function.

The algorithm implementations in cryptlib are all taken from encryption standards documents and/or reference implementations, and the library performs extensive self-tests on initialisation to ensure that a particular implementation is functioning correctly. If a module fails the self-test, its use is automatically disabled by the library. cryptlib takes great care to protect sensitive information in memory by allowing access to encryption keys and related information only through opaque "encryption contexts" which don't allow access to any of the data, erasing all keys and other critical data from memory on exit, and even locking memory pages if possible to prevent them from being swapped to disk.

[Editorial note: cryptlib is currently undergoing extensions of the certificate management functionality; capabilities may have changed by the time this paper is published]

Acknowledgments

The author would like to thank the ASB Bank for sponsoring this research, and Derek Atkins, Rolf Michelsen, Eric Young, and the readers of the Cypherpunks mailing list for feedback and comments on the paper.

References

[2600 1992] "British Credit Holes", 2600 Magazine, Summer 1993, p.12.

[ABA 1995] American Bar Association, "Digital Signature Guidelines: Legal Infrastructure for Certification Authorities and Electronic Commerce", Information Security Committee, Electronic Commerce and Information Technology Division, American Bar Association, 5 October 1995.

[AccessData] Software which will break the encryption in WordPerfect (versions 4.2-6.1, regular or enhanced encryption), Microsoft Word (versions 2.0-6.1), Microsoft Excel (all versions including the Macintosh one), Lotus 1-2-3 (all versions), Quattro Pro, Paradox, Pkzip, Norton's Diskreet (both DES and proprietary encryption), Novell NetWare (versions 3.x-4.x), and others is sold by AccessData in Utah.

[Anderson 1993] Ross Anderson, "The Legal Threat to Comsec", posting to `comp.org.eff.talk` newsgroup, message-ID `1993Aug14.103016.16342@infodev.cam.ac.uk`, 16 August 1993.

[Anderson 1994a] Ross Anderson, "Liability and Computer Security — Nine Principles", *Proceedings of ESORICS'94*, Springer-Verlag Lecture Notes in Computer Science **Vol.875**, 1994, p.231.

[Anderson 1994b] Ross Anderson, "Whither Cryptography", *Information Management and Computer Security*, **Vol.2**, **No.5** (1994), p.13.

[Anderson 1994c] Ross Anderson, "Why Cryptosystems Fail", *Proceedings of the 1993 ACM Conference on Computer and Communications Security*, p.215. Also in *Communications of the ACM*, **Vol.37**, **No.11** (November 1994), p.40, in shortened form.

[Anderson 1995] Ross Anderson and Roger Needham, "Robustness principles for public key protocols", *Advances in Cryptology — Crypto'95*, Springer-Verlag Lecture Notes in Computer Science, **Vol.963**, p.236.

[Anderson 1996a] Ross Anderson, "Important UK court case", posting to `set-discuss` mailing list, 9 July 1996.

[Anderson 1996b] Ross Anderson and Markus Kuhn, "Tamper Resistance — a Cautionary Note", *Proceedings of the Second Workshop on Electronic Commerce*, Oakland, California, November 1996.

[Anon 1994a] Anonymous, "YP-Codes", posting to `de.org.ccc newsgroup`, message-ID `199404190001.AA19957@xtropia`, 18 April 1994.

[Anon 1994b] 'David Sterndark' (an alias), "RC4 Algorithm revealed", posting to `sci.crypt newsgroup`, message-ID `sternCvKL4B.Hyy@netcom.com`, 14 September 1994.

[Anon 1994c] Anonymous, "'Alleged RC4' not real RSADSI code", posting to `sci.crypt newsgroup`, message-ID `9409250900.AA17035@ds1.tu-wien.ac.at`, 25 September 1994.

[Anon 1996a] Anonymous, "RC2 source code", posting to `sci.crypt newsgroup`, message-ID `4ehmfs$6nq@utopia.hacktic.nl`, 29 January 1996.

[Anon 1996b] Anonymous, "IPG cracked with known plaintext", posting to `cypherpunks` mailing list, message-ID `199603191732.RAA17262@pangaea.hypereality.co.uk`, 19 March 1996.

[Ante 1996] Spence Ante, "Everything You Ever Wanted To Know About Cryptography Legislation… (But Were Too Sensible to Ask)", *PC World*, May 1996.

[Apache] "Apache HTTP Server Project", `http://www.apache.org`.

[Arthur 1995] Charles Arthur, "Internet's 30bn Pound Secret Revealed", *UK Independent*, 17 August 1995.

[Atkins 1994] Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland, "The Magic Words are Squeamish Ossifrage", *Proceedings of Asiacrypt'94*, Springer-Verlag Lecture Notes in Computer Science **Vol.917**, 1995, p.263.

[Atkins 1996a] Derek Atkins, "PGP Library API", work-in-progress.

[Atkins 1996b] Derek Atkins, "PGP Library Functional Specification", work-in-progress

[Atkinson 1995a] R. Atkinson, "Security Architecture for the Internet Protocol", RFC 1825, August 1995.

[Atkinson 1995b] R. Atkinson, "IP Authentication Header", RFC 1826, August 1995.

[Atkinson 1995c] R. Atkinson, "IP Encapsulating Security Protocol (ESP)", RFC 1827, August 1995.

[BAB 1992] "Card Fraud: Banking's Boom Sector", *Banking Automation Bulletin for Europe*, March 1992, p.1.

[Back 1995a] Adam Back, "Announce: Brute force of RC4, 40 bits all swept", posting to `sci.crypt` newsgroup, message-ID `DC08Dz.LwG@exeter.ac.uk`, 20 July 1995.

[Back 1995b] Adam Back, "Another SSL breakage…",. posting to `cypherpunks` mailing list, 17 August 1995.

[Beachle 1997] Ralf Baechle, "Re: Announcement: Organisation Committee", posting to des-challenge mailing list, message-ID `199702152346.AAA08714@informatik.uni-koblenz.de`, 16 February 1997.

[BAN 1989] Michael Burrows, Martin Abadi, and Roger Needham, "A Logic of Authentication", *Proceedings of the Royal Society*, Series A, **Vol.426**, **No.1871**, 1989, p.233.

[Banker 1991] "The cost of inflexible friends", *The Banker*, January 1993, p.62.

[Barlow 1992] John Perry Barlow, "Decrypting the Puzzle Palace", *Communications of the ACM*, **Vol.35**, **No.7** (July 1992), p.25.

[Baron 1996] Andy Baron, "Microsoft IIS vv. 1.x, 2.0b New Security Bugs Alert", posting to NT Security mailing list, 30 June 1996.

[Battelle 1995] John Batelle, "Sun's Codemaking Comrades", *Wired 3.11*, p.49.

[BCS 1995] British Computer Society Code of Practice, June 1995.

[Beck 1995] Alan Beck, "Netscape's Export SSL Broken by 120 Workstations and One Student", HPCwire, 22 August 1995.

[Bellovin 1993] Steven Bellovin, "Packets Found on an Internet", *Computer Communications Review*, **Vol.23**, **No.3** (July 1993), p.26.

[Benaloh 1995] Josh Benaloh, Butler Lampson, Daniel Simon, Terence Spies, and Bennet Yee, "The Private Communication Technology Protocol (PCT)", Microsoft Corporation, October 1995.

[Bhimani 1996] Anish Bhimani, "Securing the Commercial Internet", *Communications of the ACM*, **Vol.39**, **No.6** (June 1996), p.29.

[Biham 1997] Eli Biham, "A Fast New DES Implementation in Software", *Fast Software Encryption*, Springer-Verlag Lecture Notes in Computer Science, 1997 (to appear).

[Blaze 1994] Matt Blaze, "Protocol Failure in the Escrowed Encryption Standard", *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994.

[Blaze 1995] Matt Blaze, "My life as an international arms courier", posting to `comp.org.eff.talk` newsgroup, message-ID `mab.789429326@big.att.com`, 8 January 1995.

[Blaze 1996a] Matt Blaze, Whitfield Diffie, Ronald Rivest, Bruce Schneier, Tsutomu Shimomura, Eric Thompson, and Michael Wiener, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security", A Report by an Ad Hoc Group of Cryptographers and Computer Scientists, January 1996.

[Blaze 1996b] Matt Blaze, "Re: Announcement — Warning to Crypto and Banking Communities", posting to the `coderpunks` mailing list, message-ID `199611040652.BAA13490@crypto.com`, 4 November 1996.

[Bloodmask 1996] 'Bloodmask', "Vulnerability in ALL linux distributions", posting to `linux-alert` mailing list, 30 July 1996.

[Booker 1996] Ellis Booker, "Apache Leads Web Server Pack", *Web Week*, **Vol.2**, **Issue 6** (20 May 1996).

[BorderWare 1995] "BorderWare Firewall Server Product Description", Border Network Technologies, 1995.

[Borenstein 1996] Nathaniel Borenstein et al, "Perils and Pitfalls of Practical Cybercommerce", *Communications of the ACM*, **Vol.39**, **No.6** (June 1996), p.36.

[Brooks 1995] Piete Brooks, "Cypherpunks 'brute' key cracking ring", `http://www.brute.cl.cam.ac.uk/brute/`.

[Brown 1993] P.Brown, "Digital Signatures: Can they be accepted as legal signatures in EDI?", *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 3 November 1993, p.86.

[Brown 1994] P.Brown, "Digital Signatures: Are They Legal for Electronic Commerce?", *IEEE Communications Magazine*, **Vol.32**, **No.9** (September 1994), p.76.

[BSAFE 1994] BSAFE 2.1 software, RSA Data Security Inc, 1994.

[Camp 1995] L.Jean Camp, Marvin Sirbu, and Doug Tygar, "Token and Notational Money in Electronic Commerce", Proceedings of the First USENIX Workshop on Electronic Commerce, New York, New York, July 1995, p.1.

[Cards International 1994] "High tech helps card fraud decline", *Cards International*, **No.117** (29 September 1994).

[Caronni 1997a] Germano Caronni, “Re: RC5-12/32/5 contest solved”, posting to cypherpunks mailing list, message-ID 199701291518.QAA24267@kom30.ethz.ch, 29 January 1997.

[Caronni 1997b] Germano Caronni, “Thousands of Computers Combine to Crack the Hardest Cryptographic Challenge Ever” (press release), 13 February 1997.

[CCITT 1988] “The Directory — Authentication Framework”, CCITT Recommendation X.509, 1988 (also ISO 9594-8:1988).

[CERT] “CERT Coordination Centre”, http://www.cert.org.

[CERT 1995] “CERT Summary CS-95:03”, Computer Emergency Response Team, 28 November 1995.

[CGI Security] “The Common Gateway Interface: Security”, http://hoohoo.ncsa.uiuc.edu/cgi/security.html.

[CIAC] “U.S. Department of Energy Computer Incident Advisory Capability”, http://ciac.llnl.gov.

[CIAC 1995] “Unix NCSA httpd Vulnerability”, U.S. Department of Energy Computer Incident Advisory Capability notice F-11, 14 February 1995.

[ClariNet 1997] “German Hackers Show Off Quicken Cracking Software”, Newsbytes News Network, 12 February 1997.

[Clark 1996a] Don Clark, “Researchers Find Big Security Flaw In Java Language” *The Wall Street Journal*, 26 March 1996, p.B4.

[Clark 1996b] Tim Clark, “Gates: Explorer will be huge”, *c|net*, 1 August 1996.

[Clinton 1996] William Clinton, “Executive Order: Administration of Export Controls on Encryption Products”, 15 November 1996.

[Clough 1993] B.Clough, “Cheating at Cards”, RMDP Ltd, 1993.

[Coe 1995] Diane Coe and Judith Furlong, “Developing and Deploying Corporate Cryptographic Systems”, Proceedings of the First USENIX Workshop on Electronic Commerce, New York, New York, July 1995, p.137.

[Collins 1996] Lyal Collins, “Cybank breaks new ground; rejects public-key encryption”, posting to cypherpunks mailing list, message-ID 31E9DCF8.43EA@ozemail.com.au, 14 July 1996.

[Consultant 1996] ‘Consultant’, “Excellent employment in cryptrography [sic]”, posting to sci.crypt newsgroup, message-ID DvDG4D.8pD@linex1.linex.com, 30 July 1996.

[Cooper 1996] Russ Cooper, “Warning — WebConsole opens DoS hole in NT”, posting to NT Security mailing list, 18 February 1996.

[Corcoran 1996] Elizabeth Corcoran, “Scrambling for a Policy on Encryption Exports”, *The Washington Post*, 25 February 1996, p.H1.

[COST] COST, http://www.cost.se.

[Cowrie 1996] J Cowie, B Dodson, Arjen Lenstra, Peter Montgomery, et al, “NFS factoring”, Proceedings of Crypto’96.

[CRAK Software] Software to break a variety of popular word processor, spreadsheet, and financial programs including MS Excel 5.0, Lotus 123 version 4.0, Quattro Pro 6.0, MS Word 6.0, Wordperfect through to version 5.2, and Quicken through to version 4.0 is sold by CRAK Software.

[Crash Netscape] http://www.comland.com/~cult/trust.html (this page is difficult to reference properly because it crashes Netscape, and occasionally the entire machine, when connected to).

[cryptlib] cryptlib Information, http://www.cs.auckland.ac.nz/~pgut001/cryptlib.html.

[CSPP 1996] Computer Systems Policy Project, “Perspectives on Security in the Information Age”, January 1996.

[Ctrl2Cap] “Ctrl2Cap — Caps-lock to Ctrl Mapper for Windows NT”, http://www.ntinternals.com/ctrl2cap.htm.

[Curtin 1996] C.Matthew Curtin, “Snake-Oil Warning Signs: Encryption Software to Avoid” (the Snake Oil FAQ), version 1.1, November 1996.

[Daily Planet 1997] “Hackers Kraken Bankrekening”, Daily Planet, 7 February 1997.

[DataFellows] “F-Secure Cryptography Products”, http://www.europe.datafellows/com/f-secure/.

[Datenschutzberater 1994] “BDSG und #203 Strafgesetzbuch”, *Datenschutzberater*, 15 November 1994, p.1.

[dcc] “The dcc decompiler”, http://www.cs.uq.edu.au/groups/csm/dcc.html.

[Dean 1996] Drew Dean, Edward Felten, and Dan Wallach, “Java Security: From HotJava to Netscape and Beyond”, *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, IEEE Press, 1996.

[DEPL] ftp://ftp.giga.or.at/pub/hacking/dos/depl.zip.

[Diffie 1977] Whitfield Diffie and Martin Hellman, “Exhaustive Cryptanalysis of the NBS Data Encryption Standard”, *IEEE Computer*, **Vol.10**, **No.6** (June 1977), p.74.

[Diffie 1992] Whitfield Diffie, Paul van Oorschot, and Michael Wiener, “Authentication and Authenticated Key Exchanges”, *Designs, Codes, and Cryptography*, **Vol.2**, 1992, p.107.

[Dittes 1994] Axel Dittes, “Testballon geplatzt: Chaos Computer Club knackt Vershlüsselung von Software-CD”, *c’t*, May 1994, p.19.

[DOC 1980] Department of Commerce, “White Paper of National Security Policy Options for Cryptography”, National Communications and Information Administration, 17 November 1980.

[DOC 1995] “A Study of the International Market for Computer Software with Encryption”, Department of Commerce and National Security Agency, prepared for the Interagency Working Group on Encryption and Telecommunications Policy, 1995.

[Doligez 1995a] Damien Doligez, “SSL challenge — broken!”, posting to sci.crypt newsgroup, message-ID 40sajr$sps@news-rocq.inria.fr, 16 August 1995.

[Doligez 1995b] Damien Doligez, “SSL challenge virtual press conference”, http://pauillac.inria.fr/~doligez/ssl/press-conf.html.

[DOS 1989] Department of State, “International Traffic in Arms Regulations” (ITAR), 22 CFR 120-130, Office of Munitions Control, November 1989.

[DOS 1992] Department of State, “Defense Trade Regulations” (DTR), 22 CFR 120-130, Office of Munitions Control, May 1992.

[Dunlap 1996] Charlotte Dunlap and Deborah Gage, “Channel feels pinch of export limitations — VARs Hit Encryption Roadblock: Could 40 bits of code cost you that multimillion-dollar bid?”, *Computer Reseller News*, 5 August 1996, p.51.

[Eastlake 1994] Donald Eastlake, Stephen Crocker, Jeffrey Schiller, “Randomness Recommendations for Security”, RFC 1750, December 1994.

[EuroSign] EuroSign — The European Certification Authority, http://eurosign.com.

[Farmer 1996] Dan Farmer, "Shall we Dust Moscow?", `http://www.trouble.org/survey/`, 18 December 1996.

[Federal Register 1996] Federal Register, Public Notice 2294, Vol.61, No.33, 16 February 1996, p.6111.

[Felten 1996a] Ed Felten, "Security Flaw in Netscape 2.02", *Risks-Forum Digest* **Vol.18**, **Issue 13**, 17 May 1996.

[Felten 1996b] Ed Felten, "Java security update", *Risks-Forum Digest*, **Vol.18**, **Issue 32**, 13 August 1996.

[Felten 1996c] Ed Felten "Internet Explorer Security Problem", *Risks-Forum Digest*, **Vol.18**, **Issue 36**, 21 August 1996.

[Finney 1995a] Hal Finney, "SSL RC4 Challenge", posting to `sci.crypt newsgroup`, message-ID `3u6kmg$pm4@jobe.shell.portal.com`, 14 July 1995.

[Finney 1995b] Hal Finney, "SSL Challenge #2", posting to `cypherpunks` mailing list, message-ID `199508191525.IAA16294@jobe.shell.portal.com`, 19 August 1995.

[FIPS 1980] "DES Modes of Operation", FIPS Publication 81, 1980.

[FIRST] "Forum of Incident Response and Security Teams", `http://www.first.org`.

[Florida 1996] "Electronic Signature Act of 1996", Florida SB 942, 31 May 1996.

[Floyd 1997] J.Floyd, "Distributed Search Management Protocol", Internet RFC draft, February 1997.

[Fox 1996] Robert Fox, "News Track", *Communications of the ACM*, **Vol.39**, **No.10** (October 1996), p.9.

[funet] `ftp://ftp.funet.fi/pub/crypt/cryptography/other`.

[Garfinkel 1994] Simson Garfinkel, "PGP — Pretty Good Privacy", O'Reilly and Associates, 1994.

[Gauthier 1995] Paul Gauthier, "Basic Flaws in Internet Security and Commerce", posting to `comp.security.unix newsgroup`, message-ID `gauthier.813274073@espresso.CS.Berkeley.EDU`, 9 October 1995.

[Geary 1994] J.Geary, "Executive Liability for Computer Crime and How to Prevent It", *Information Management and Computer Security*, **Vol.2**, **No.2**, 1994, p.29.

[Georgia 1996] "Georgia Digital Signature Act (Draft)", Georgia Digital Signature Task Force, 23 February 1996.

[Goldberg 1995] Ian Goldberg, "Netscape SSL implementation cracked!", posting to `cypherpunks` mailing list, message-ID `199509180441.VAA16683@lagos.CS.Berkeley.EDU`, 18 September 1995.

[Goldberg 1996a] Ian Goldberg and David Wagner, "Randomness and the Netscape Browser", *Dr.Dobbs Journal*, January 1996, p.66.

[Goldberg 1996b] Ian Goldberg and David Wagner, "Architectural Considerations for Cryptanalytic Hardware", `http://www.cs.berkeley.edu/~iang/isaac/hardware/`.

[Goldberg 1997] Ian Goldberg, "Exportable Cryptography Totally Insecure: Challenge Cipher Broken Immediately" (press release), 28 January 1997.

[Gordon 1994] Sara Gordon, "Technology Enables Crime", *Proceedings of IFIP 1994 Security Conference*, May 1994.

[Green 1995] Heather Green, "Netscape Says Hackers Uncover 3rd Flaw in Its Internet Software", *The New York Times*, 25 September 1995.

[Gutmann 1995] Peter Gutmann, "How Windows encrypts .PWL files", posting to comp.security.misc newsgroup, message-ID 49f1u0$6ai@net.auckland.ac.nz, 28 November 1995.

[Gutmann 1996a] Peter Gutmann, "Specification for Ron Rivests Cipher No.2", posting to sci.crypt newsgroup, message-ID 4fk39f$f70@net.auckland.ac.nz, 11 February 1996.

[Gutmann 1996b] Peter Gutmann, "Subliminal channel via PKCS #1", posting to sci.crypt.research newsgroup, message-ID pgpmoose.199602111222.44767@paganini.sydney.sterling.com, 11 February 1996.

[Gutmann 1996c] Peter Gutmann and Jenny Shearer, "Government, Cryptography, and the Right to Privacy", *Journal of Universal Computer Science*, **Vol.2**, **No.3** (28 March 1996), p.113.

[Gutmann 1996d] Peter Gutmann, "How to break Netscape's server key encryption", posting to the cypherpunks mailing list, message-ID 84366802803808@cs26.cs.auckland.ac.nz, 25 September 1996.

[Gutmann 1996e] Peter Gutmann, "How to break Netscape's server key encryption — Followup", posting to the cypherpunks mailing list, message-ID 84373168812186@cs26.cs.auckland.ac.nz, 26 September 1996.

[hacktic] ftp://utopia.hacktic.nl/pub/replay/cracking/.

[Hager 1996] Nicky Hager, "Secret Power", Craig Potton Publishing, 1996.

[Hallam-Baker 1996] Phill Hallam-Baker, "A problem with Navigator's cache — Reply", posting to www-security mailing list, 25 August 1996.

[Harriman 1990] D.Harriman, "Password Fishing on Public Terminals", *Computer Fraud and Security Bulletin*, January 1990, p.12.

[Head 1995] B.Head, "Playing the Game", *Banking Technology*, June 1995, p.24.

[Heinlein 1993] "United States information systems security — a myth?", *Computer Fraud and Security Bulletin*, May 1993, p.13.

[Hellman 1979] Martin Hellman, "DES will be totally insecure within 10 years", *IEEE Spectrum*, **Vol.16**, **No.7** (July 1979), p.32.

[Hinden 1996] Robert Hinden, "IP Next Generation Overview", *Communications of the ACM*, **Vol.39**, **No.6** (June 1996), p.61.

[Hoffman 1994] Lance Hoffman, Faraz Ali, Steven Heckler, and Ann Huybrechts, "Cryptography Policy", *Communications of the ACM*, **Vol.37**, **No.9** (September 1994), p.109.

[Hoffman 1995] Lance Hoffman (ed), "Building in Big Brother: The Cryptographic Policy Debate", Springer-Verlag, 1995.

[Hopwood 1996] David Hopwood, "Another Java security bug", posting to comp.lang.java newsgroup, message-ID 4orf1q$t6f@news.ox.ac.uk, 2 June 1996.

[Horgan 1985] John Horgan, "Thwarting the information thieves", *IEEE Spectrum*, July 1985, p.30.

[Hyde 1976] Edward Hyde, "The Phone Book", Henry Regnery Company, 1976.

[ICE 1996] Integrated Computing Engines, "MIT Student Uses ICE Graphics Computer To Break Netscape Security in Less Than 8 Days", 10 January 1996.

[IEEE 1995] IEEE P1363 Appendix E, "Cryptographic Random Numbers", Draft version 1.0, 11 November 1995.

[InfoWorld 1996] "Security is not just skin-deep", *InfoWorld*, 16 January 1996.

[International StrongHold] "International Stronghold Homepage", http://stronghold.ukweb.com/.

[ISO 1987] "Information Technology — Modes of Operation for a 64-bit Block Cipher", ISO 8372:1987, 1987.

[ISO 1991a] "Information Technology — Security Techniques — Modes of Operation for an n-bit Block Cipher Algorithm", ISO 10116:1991, 1991.

[ISO 1991b] "Banking and Financial Related Services — Procedures for Message Encipherment (Wholesale)", ISO 10126-2:1991, 1991.

[ISO 1994a] "Banking — Key management by means of asymmetric algorithms", ISO 11166:1994, 1994.

[ISO 1994b] "Information Technology — Security Techniques — Key Management", ISO DIS 11770:1994, 1994.

[ITSEC 1992] Commission of the European Communities, "Information Technology Security Evaluation Criteria (ITSEC)", Brussels, 1991.

[Java Support] "Updating the Java Support on a User's Machine", http://www.microsoft.com/java/sdk/getstart/javac007.htm.

[Johnson 1993] D.Johnson, M.Matyas, A.Le, and J.Wilkins, "Design of the commercial data masking facility data privacy algorithm", *First ACM Conference on Computer and Communications Security*, ACM Press, 1993, p.93.

[Kahaner 1996] Larry Kahaner, "Competitive Intelligence: From Black Ops to Boardrooms — How Businesses Gather, Analyze, and use Information to Succeed in the Global Marketplace", Simon and Schuster, 1996.

[Karas 1997] Brian Karas, "MSIE/Netscape Java Hacks", posting to com-priv mailing list, message-ID 199702261350.AA16194@internet-mail2.ford.com, 26 February 1997.

[Karn 1995] Phil Karn, Perry Metzger, and W. Simpson, "The ESP DES-CBC Transform", RFC 1829, August 1995.

[Kaufman 1995] Charlie Kaufman, Radia Perlman, and Mike Speciner, "Network Security: PRIVATE Communication in a PUBLIC World", Prentice-Hall, 1995.

[KBDC] ftp://ftp.simtel.net/pub/simtelnet/msdos/sysutl/kbdcv10.zip.

[Kennedy 1996] David Kennedy, "Another Netscape Bug US$1K", *Risks-Forum Digest*, **Vol.18**, **Issue 1422**, May 1996.

[KeyCopy] ftp://ftp.clark.net/pub/jcase/keycopy.zip.

[KeyTrap] http://frosted.mhv.net/keytrap.html.

[Klaus 1995] Christopher Klaus, "Stealth Scanning — Bypassing Firewalls/SATAN Detectors", posting to ISS security alert mailing list, message-ID 199512020131.UAA15017@iss.net, 1 December 1995.

[Klaus 1996] Christopher Klaus, "MicroSoft Web Exploit", posting to NT Security mailing list, 1 July 1996.

[Kochanski 1987] Martin Kochanski, "A Survey of Data Insecurity Packages", *Cryptologia*, **Vol.11**, **No.1** (January 1987), p.1.

[Krieger 1997] Todd Krieger, "Hackers Go on TV to Show Perils in ActiveX", New York Times, 13 February 1997.

[Kuner] Christopher Kuner, "Law of Electronic and Internet Commerce in Germany", http://ourworld.compuserve.com/homepages/ckuner/.

[LeBlanc 1996a] David LeBlanc, “rshsvc from Resource Kit”, posting to NT Security mailing list, 2 May 1996.

[LeBlanc 1996b] David LeBlanc, “NT-ftp server”, posting to NT Security mailing list, 31 May 1996.

[Lenstra 1996] Arjen Lenstra, “New factorization record”, posting to `sci.crypt newsgroup`, message-ID `4kqkd8$g37@net.auckland.ac.nz`, 14 April 1996.

[Lewis 1996] Tony Lewis, “Reference code availability update”, posting to `set-discuss` mailing list, 31 October 1996.

[Levy 1996] Steven Levy, “Scared Bitless”, *Newsweek*, 10 June 1996, p.49.

[Leyland 1995] Paul Leyland, “The BlackNet 384-bit PGP key has been BROKEN”, posting to `alt.security.pgp` newsgroup, message-ID `PCL.95Jun26110915@sable.ox.ac.uk`, 26 June 1995.

[Lindstrom 1996] Mattias Lindstrom, “Re: MicroSoft Web Exploit”, posting to NT Security mailing list, 4 July 1996.

[Lopatic 1995] Thomas Lopatic, “Vulnerability in NCSA HTTPD 1.3”, posting to `bugtraq` mailing list, 13 February 1995

[LoVerso] John LoVerso, “JavaScript Problems I’ve Discovered”, `http://www.osf.org/~loverso/javascript/`.

[Luckhardt 1996] Norbert Luckhardt and Harald Bögeholz, “Slüsselerlebnisse: PGP-Frontends im Überblick”, *c’t*, January 1996, p.132.

[Madsen 1994] W.Madsen, “Online industrial espionage”, *Network Security*, November 1994, p.14.

[Madsen 1995] W.Madsen, “Puzzle palace conducting Internet surveillance”, *Computer Fraud and Security Bulletin*, June 1995, p.12.

[Majer 1996] Alan Majer, “mbanx — 128 bit SSL transactions”, posting to e$ mailing list, 1 November 1996.

[Marion 1991] P.Marion, “La Mission Impossible à la tête des Services Secrets”, Calmann-Lévy, France, 1991.

[Markoff 1995] John Markoff, “Security Flaw Is Discovered In Software Used in Shopping”, *The New York Times*, 19 September 1995, p.A1.

[Markoff 1996a] John Markoff, “In Search Of Computer Security”, *New York Times* special section “Business World Outlook ‘96”, 2 January 1996, p.C15.

[Markoff 1996b] John Markoff, “New Netscape Software Flaw Is Discovered”, *The New York Times*, 18 May 1996, p.31.

[Markoff 1996c] John Markoff, “White House Challenged on Data Security”, *The New York Times*, 31 May 1996, p.D2.

[Martin 1996] David Martin, “Java and Firewalls”, `http://www.cs.bu.edu/techreports/96-026-java-firewalls.ps.Z`.

[McGraw 1997] Gary McGraw, “’New’ Java Holes”, posting to `secure-mobile-code` mailing list, message-ID `199702272134.QAA05431@rstcorp.com`, 27 February 1997.

[McLain 1996] Fred McLain, “ActiveX, or how to put nuclear bombs in web pages”, `http://www.halcyon.com/mclain/ActiveX/`.

[MDR 1997] “ARD-Wirtschaftsmagazin ‘Plusminus’: Neue Sicherheitslücke beim Online-Banking”, MDR broadcast, 28 January 1997, 9:35 pm.

[Merriman 1996] George Merriman, “Re: Guest account enabled on server?”, posting to NT Security mailing list, 20 June 1996.

[Metzger 1995] Perry Metzger and W. Simpson, “IP Authentication using Keyed MD5”, RFC 1828, August 1995.

[Mitre 1977] Mitre Corporation, “Vulnerability of Electronic Communications Systems to Electronic Interception”, 1977.

[McCullagh 1994] E.McCullagh and I.Ryan, “Who pays the bills”, *Cards International*, **No.108** (25 April 1994), p.8.

[McGraw 1997] Gary McGraw and Edward Felten, “Java Security: Hostile Applets, Holes, and Antidotes — What Every Netscape and Internet Explorer User Needs to Know”, John Wiley and Sons, New York, 1997.

[McLeod 1993] K. McLeod, “Covert Local Area Network Surveillance”, *Computer Audit Update*, October 1993, p.13.

[Microsoft 1996] “Microsoft CryptoAPI Application Programmers Guide”, Version 1 (Preliminary), July 1996.

[Milberg 1995] Sandra Milberg, Sandra Burke, Jeff Smith, and Ernest Kallman, “Values, Personal Information, Privacy, and Regulatory Approaches”, *Communications of the ACM*, **Vol.38**, **No.12** (December 1995), p.65.

[Mueller 1996] Marianne Mueller, “Java security”, *Risks-Forum Digest*, **Vol.17**, **Issue 79**, 23 February 1996.

[Munger 1995] Benoit Munger, “Cachez ces mots que je ne saurais lire”, *Le Devoir*, 28 August 1995.

[NCSA 1995] NCSA httpd 1.4.1 release notes, May 1995.

[Netcraft] “Netcraft — Web Server Survey”, `http://www.netcraft.co.uk/Survey/Reports/`.

[Netwatcher 1996] “NetWatcher (New Product)”, posting to `comp.security.unix` newsgroup, message-ID `4g84eg$fvr@kirk.nrv.net`, 18 February 1996.

[Neumann 1995] Peter Neumann, “Third Netscape weakness found”, *Risks-Forum Digest*, **Vol.17**, **Issue 36**, 27 September 1995.

[Neumann 1996a] Peter Neumann, “More Java, JavaScript, and Netscape problems”, in “Security Risks in Computer-Communication Systems”, *ACM SIGSAC Review*, **Vol.14**, **No.3** (July 1996), p.22.

[Neumann 1996b] Peter Neumann, “Risks in Digital Commerce”, Inside Risks, *Communications of the ACM*, **Vol.39**, **No.1** (January 1996), p.154.

[NewAPI32] `ftp://ftp.simtel.net/pub/simtelnet/win3/prog/newapi32.zip`.

[NewsBytes 1995] “Netscape Encrypted Data Cracked”, *NewsBytes*, Tokyo, Japan, 18 August 1995.

[NIST 1994] National Institute of Standards and Technology, “Digital Signature Standard”, FIPS Publication 186, May 1994.

[Norton 1994] M.Norton, “Close to the nerve”, *Banking Technology*, December 1994, p.29.

[NRC 1996] “Cryptography’s Role in Securing the Information Society”, Committee to Study National Cryptography Policy, Computer Science and Telecommunications Board, National Research Council, National Academy of Science and National Academy of Engineering, 30 May 1996.

[NYT 1995a] “How a computer sleuth traced a digital trail”, *The New York Times*, 15 February 1995.

[NYT 1995b] “Coding-Export Limits Opposed”, *The New York Times*, 9 November 1995, p.D4.

[NYT 1996] "Web browser classified as weapon", *The New York Times*, 7 November 1996.

[Odlyzko 1996] Andrew Odlyzko, "The Future of Integer Factoring", *CryptoBytes*, RSA Data Security, 1996.

[Password spoof] `http://blind.trust.cs.cmu.edu/spoof.class`.

[Password Thief] The Password Thief for Windows 95, `http://ourworld.compuserve.com/homepages/esmsoftware/Thief.htm`.

[PC Week 1996] "Programmer finds flaw in Windows NT", PC Week Online, 20 December 1996.

[PersonalSecure] "The Internet Solution PersonalSecure™ Web Proxy Solution", `http://www.security.is.co.za/Pages/HTTPPersonalSecure.htm`.

[Peter 1997] Steffen Peter and Lutz Donnerhacke, "Geldklau dank ActiveX", *iX* 3/97, March 1997.

[Pettitt 1997] John Pettitt, "ActiveX — a real world view", *Risks-Forum Digest*, **Vol.18**, **Issue 84**, 22 February 1997.

[PGP] "MIT distribution site for PGP", `http://web.mit.edu/network/pgp.html`.

[Phantom] `ftp://wuarchive.wustl.edu/pub/msdos/keyboard/ptm229i.zip`.

[Phillips] Paul Phillips, "The CGI Security FAQ", `http://www.primus.com/staff/paulp/cgi-security/`.

[Pietrek 1994a] Matt Pietrek, "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format", *Microsoft Systems Journal*, March 1994.

[Pietrek 1994b] Matt Pietrek, "Learn System-Level Win32 Coding Techniques by Writing an API Spy Program", *Microsoft Systems Journal*, December 1994.

[Piper 1993] F.Piper and N.Stephens, "Digital Signatures: RSA or El Gamal?", *Proceedings of the Third Conference on Cryptography and Coding*, England, 1991, p.311.

[Pizzo 1996] Stephen Pizzo and David Sims, "First Virtual's 'Killer-App' — Company Says 'Switch to Us and No One Gets Hurt'", *GNN Web Review*, 30 January 1996.

[Plumb 1994] Colin Plumb, "Truly Random Numbers", *Dr.Dobbs Journal*, November 1994, p.113.

[Plumb 1995] Colin Plumb, "Re: My life as an international arms courier", posting to `comp.org.eff.talk` newsgroup, message-ID `3el9cm$7og@nyx10.cs.du.edu`, 8 January 1995.

[Prettejohn 1996] Mike Prettejohn, "Over 10,000 sites running nonsecure versions of NCSA web server", *Risks-Forum Digest*, **Vol.17**, **Issue 88**, 11 March 1996.

[r³] "r³ SSL HTTP Solution", `http://www.r3.ch/products/index.html`

[Racal 1988] "GSM System Security Study", Racal Research Ltd, 10 June 1988.

[Ramos 1996] Frank Ramos, "Re: Guest account enabled on server?", posting to NT Security mailing list, 19 June 1996.

[Ranum] Marcus J.Ranum, "A Taxonomy of Internet Attacks", `http://www.clark.net/pub/mjr/pubs/attck/index.htm`.

[Ranum 1996] Marcus Ranum, "Firewalls: Are they being used right? Are they cost-effective?", invited talk, 6th USENIX Security Symposium, San Jose, California, July 1996.

[Raymond 1991] Eric Raymond, "The New Hackers Dictionary", MIT Press, 1991.

[ravemaster 1994] 'ravemaster', "Neue YP Codes", posting to de.org.ccc newsgroup, message-ID 2q63qd$huf@winx03.informatik.uni-wuerzburg.de, 3 May 1994.

[Reif 1995a] Holger Reif, "Netz ohne Angst: Sicherheitsrisiken des Internet", *c't*, September 1995, p.174.

[Reif 1995b] Holger Reif, "Peinliche Panne: Netscape gibt ernsthafte Sicherheitslücken zu", *c't*, November 1995, p.26.

[Reif 1996] Holger Reif, "Schlüsselfertig — Secure Socket Layer: Chiffrieren und Zertifieren mit SSLeay", *iX*, **No.6/96** (July 1996), p.128.

[Rescorla 1994] Eric Rescorla, "RC4 compatibility testing", posting to cypherpunks mailing list, message-ID 9409140137.AA17743@eitech.eit.com, 13 September 1994.

[Reuter 1995] "Clinton instructs CIA to focus on trade espionage", Reuter, Los Angeles, 23 July 1995.

[Richter 1994] Jeffrey Richter, "Load Your 32-bit DLL into Another Processes Address Space Using INJLIB", *Microsoft Systems Journal*, May 1994.

[RingZero 1995] 'RingZero', "NEW Netscape RNG hole", posting to cypherpunks mailing list, message-ID 9510080732.AA14015@anon.penet.fi, 8 October 1995.

[Robertson 1995] Richard Robertson, "Random Number Generators Draft", IEEE P1363 Random Number Generators Review Group, May 1995.

[Robertson 1996] Jack Robertson, "Code Limit Exceeded", *Electronic Buyers' News*, **Issue 1030**, 28 October 1996

[Roos 1995] Andrew Roos, "A Netscape Server implementation error", posting to cypherpunks mailing list, message-ID 305F486F@beetle.vironix.co.za, 19 September 1995.

[Root 1991] W. Root, "United States Export Controls, 3rd ed.", Prentice Hall Law and Business, Englewood Cliffs, NJ, 1991.

[Ross 1996] David Ross, "I decrypted Windows 95 share passwords…", posting to the cypherpunks mailing list, message-ID 199602111738.MAA15947@bb.hks.net, 11 February 1996.

[Rothke 1996] Ben Rothke, "Exploding the C2 myth: C2-level security had its place in the past, but today's systems need security that is much more robust", *InfoWorld*, **Vol.18**, **Issue 35** (26 August 1996).

[RSADSI 1997] RSA Data Security Inc, "RSA Announces New 'Des Challenge'" (press release), 2 January 1997.

[RSA Labs 1995] "Answers to Frequently Asked Questions about Today's Cryptography", Version 3.0, RSA Laboratories, 1995.

[Rueppel 1993] Rainer Rueppel, "Criticism of ISO CD 11166 Banking: Key Management by Means of Asymmetric Algorithms", *Proceedings of the 3rd Symposium on the State and Progress of Research in Cryptography*, Rome, February 1993, p.191.

[Russinovich 1997] Mark Russinovich and Bryce Cogswell, "Windows NT System-Call Hooking", *Dr.Dobbs Journal*, January 1997, p.42.

[SafePassage] "SafePassage Web Proxy", http://www2.c2.net/products/spwp.

[Sandberg 1995a] Jared Sandberg, "French Hacker Cracks Netscape Code, Shrugging Off U.S. Encryption Scheme", *The Wall Street Journal*, 17 August 1995, p.B3.

[Sandberg 1995b] Jared Sandberg, "Netscape's Internet Software Contains Flaw That Jeopardizes Security of Data", *The Wall Street Journal*, 19 September 1995.

[Sandberg 1995c] Jared Sandberg, "Netscape Software for Cruising Internet Is Found to Have Another Security Flaw", *The Wall Street Journal*, 25 September 1995, p.B12.

[Schreiber 1997] Sven Schreiber, "A Spy Filter Driver for NT", *Windows Developer's Journal*, February 1997.

[Scriba 1997] Jürgen Scriba, "Computer: Schleusen geöffnet", *Der Spiegel*, 8 January 1997.

[Segev 1996] Arie Segev, Jaana Porra, and Malu Roidan, "Financial EDI Over the Internet, Case Study II: The Bank of America and Lawrence Livermore National Laboratory Pilot", Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, California, November 1996, p.173.

[Seminerio 1997] Maria Seminario, "Hackers claim ActiveX code can be used to pilfer funds online", *PC Week*, 6 January 1997.

[Simmons 1993a] Gus Simmons, "The subliminal channel in the US digital signature algorithm (DSA)", *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography, 1993*, p.35.

[Simmons 1993b] Gus Simmons, "Subliminal communications is possible using the DSA", *Proceedings of Eurocrypt '93*, Springer-Verlag Lecture Notes in Computer Science, **No.765**, 1994, p.218.

[Simmons 1994] Gus Simmons, "Cryptanalysis and Protocol Failures", *Communications of the ACM*, **Vol.37**, **No.11** (November 1994), p.56.

[Sioux] "Sioux: Sophisticated and Secure", `http://www.thawte.com/products/sioux`.

[Smith 1996] Richard Smith, "ActiveX security woes", posting to `comp.security.misc newsgroup`, message-ID `01bbc911$9ea41100$a78103c7@tiac.net.tiac.net`, 2 November 1996.

[Spafford 1988] Gene Spafford, "The Internet Worm: Crisis and Aftermath", *Communications of the ACM*, **Vol.32**, **No.6** (June 1989), p.678.

[Spiegel 1994] "Software — Bequem wie nie", *Der Spiegel*, **No.32**, 1994.

[SSL 1996] "The SSL Protocol", Version 3.0, Netscape Communications Corporation, March 1996.

[SSLeay] SSLeay and SSLapps FAQ, `http://www.psy.uq.oz.au/~ftp/Crypto/`.

[SSR] "Secure Socket Relay", `http://www.medcom.se/ssr/`.

[Stallings 1994] William Stallings, "Protect Your Privacy — A Guide for PGP Users", Prentice-Hall, 1994.

[Stein] Lincoln Stein, "The World Wide Web Security FAQ", `http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html`.

[Stevens 1994] Richard Stevens, "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1994.

[Stevens 1996] Richard Stevens, "TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and Unix Domain Protocols", Addison-Wesley, 1996.

[StrongHold] "Stronghold Homepage", `http://apachessl.c2.org`.

[Surveillance Agent] `ftp://ftp.simtel.net/pub/simtelnet/win95/security/spy1132.zip`.

[Tagesspiegel 1997] "Geldklau Ohne PIN/TAN", 29 January 1997, *Berliner Tagesspiegel*.

[Taylor 1996] Dave Taylor, "The Webmaster: Web Site Memory with Cookies", *;login*, **Vol.21**, **No.5** (October 1996), p.32.

[TCSEC 1985] Department of Defence, "Trusted Computer System Evaluation Criteria", DOD 5200.28-STD, National Computer Security Centre, December 1985.

[Thawte] Thawte Consulting, `http://www.thawte.com/certs/`.

[TR45.3 1992] "Dual-Mode Cellular System Authentication, Message Encryption, Voice Privacy Mask Generation, Shared Secret Data Generation, A-Key Verification, and Test Data", TR 45.3 (Appendix A to IS-54), Rev.B, February 1992.

[Trei 1995] Peter Trei, "Netscape's SSL security has been compromised", posting to `comp.security.misc newsgroup`, message-ID `40t4ti$b8s@iii1.iii.net`, 16 August 1995.

[Trei 1996] Peter Trei, "Optimizing DES Key Recovery in Software", posting to the `coderpunks` mailing list, message-ID `199610171918.MAA23054@toad.com`, 17 October 1996.

[Tygar 1996] Doug Tygar and Alma Whitten, "WWW Electronic Commerce and Trojan Horses", Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, California, November 1996, p.243.

[Uehling 1994] Mark Uehling, "Cracking the Uncrackable Code", *Popular Science*, September 1994.

[Uhlig 1995] Robert Uhlig, "Security threat to Internet shopping", *Daily Telegraph* (paper edition), 3 October 1995, p.12.

[unssl 1995] `ftp://ftp.csua.berkeley.edu/pub/cypherpunks/cryptanalysis/unssl.c`.

[USA Today 1995] USA Today editorial, 24 October 1995, p.12A.

[Verisign] Verisign, `http://www.verisign.com`.

[Vincent-Carrefour 1996] Jacques Vincent-Carrefour, "Autorisation de fourniture et d'utilisation generale de moyens de cryptologie No.2500", 509/DISSI dossier numero 950038, 7 November 1995.

[Vogelheim 1996] Daniel Vogelheim, "RRC.2 implementation available", posting to `sci.crypt newsgroup`, message-ID `4g5u20$e4k@news.rwth-aachen.de`, 19 February 1996.

[W4W] `ftp://ftp.funet.fi/pub/crypt/analysis/wordunp.zip`.

[Wagner 1996] David Wagner and Bruce Schneier, "Analysis of the SSL 3.0 Protocol", Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, California, November 1996, p.29.

[Walsh 1996] Mike Walsh, "Microsoft's warning", *Risks-Forum Digest*, **Vol.18**, **Issue 38**, 26 August 1996

[Washington 1996] "Digital Signature Act of 1996", Washington HB 2635, 6 February 1996.

[Wayner 1994] M.Wayner, "New Products to Shore up the Net", *Open Systems Today*, **No.156** (15 August 1994), p.8.

[Wiener 1993] Michael Wiener, "Efficient DES Key Search", Technical Report TR-244, Carleton University, 20 August 1993.

[Williams 1996] Eric Williams, "New Netscape 2.0/2.01 Security Issue (Java Sockets)", posting to `comp.lang.java newsgroup`, message-ID `4jppdt$ake@sky.net`, 1 April 1996.

[Wingfield 1996a] Nick Wingfield, "Netscape fixes Apache in its sights", *c|net*, 6 June 1996.

[Wingfield 1996b] Nick Wingfield, “Program compromises IE security”, *c|net*, 23 September 1996.

[Wingfield 1997] Nick Wingfield, “ActiveX used as hacking tool”, *c|net*, 7 February 1997.

[Wink 1996] ‘Wink Junior’, “Internet Privacy Guaranteed ad (POTP Jr)”, posting to `cypherpunks` mailing list, message-ID `199602190320.TAA15431@julie.teleport.com`, 19 February 1996.

[Wirbel 1996] Loring Wirbel, “State Dept. Tries To Quash API’s for PGP cryptography”, *Electronic Engineering Times*, 29 April 1996, p.4

[Woody 1994a] ‘Woody’, “Neue YP Codes”, posting to `de.org.ccc` newsgroup, message-ID `2q2ip5$7nc@winx03.informatik.uni-wuerzburg.de`, 2 May 1994.

[Woody 1994b] ‘Woody’, “Neue YP Codes”, posting to `de.org.ccc` newsgroup, message-ID `2q58nf$nc@winx03.informatik.uni-wuerzburg.de`, 3 May 1994.

[Woody 1994c] ‘Woody’, “Alle Yellow Point Codes!”, posting to `de.org.ccc` newsgroup, message-ID `2qnljq$36i@winx03.informatik.uni-wuerzburg.de`, 10 May 1994.

[WordPerfect] `ftp://garbo.uwasa.fi/pc/util/wppass2.zip`.

[Wright 1995] Gary Wright and Richard Stevens, “TCP/IP Illustrated, Volume 2: The Implementations”, Addison-Wesley, 1995.

[Wright 1996a] Ben Wright, “The Law of Electronic Commerce: EDI, Fax, and email (1996 edition)”, Little, Brown, 1996 (this book is updated continuously to reflect new laws and regulations covering the field).

[Wright 1996b] Ben Wright, “Legal Signatures and Proof in Electronic Commerce”, Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, California, November 1996, p.67.

[WSJ 1996] “Princeton Team Finds Bug In Part Of Netscape Program”, *The Wall Street Journal*, 20 May 1996.

[X*Presso] X*Presso Security Package Product Information, `http://www.brokat.de/xpresso/xpdprine.htm`.

[Young 1996a] Adam Young and Moti Yung, “The Dark Side of Black-Box Cryptography or Should We Trust Capstone?”, *Proceedings of Crypto’96*, Springer-Verlag Lecture Notes in Computer Science, 1996.

[Young 1996b] Eric Young, “Bank transactions on Internet”, posting to `cypherpunks` mailing list, message-ID `Pine.SOL.3.91.960409104403.28771C-100000@orb`, 9 April 1996.

[Ziff-Davis 1997] “German Hacking Group Cracks Quicken Software”, Ziff-Davis Newswatch, 13 February 1997.

[Zimmermann 1995a] Phil Zimmermann, “The Official PGP Users Guide”, MIT Press, 1995.

[Zimmermann 1995b] Phil Zimmermann, “PGP Source Code and Internals”, MIT Press, 1995.