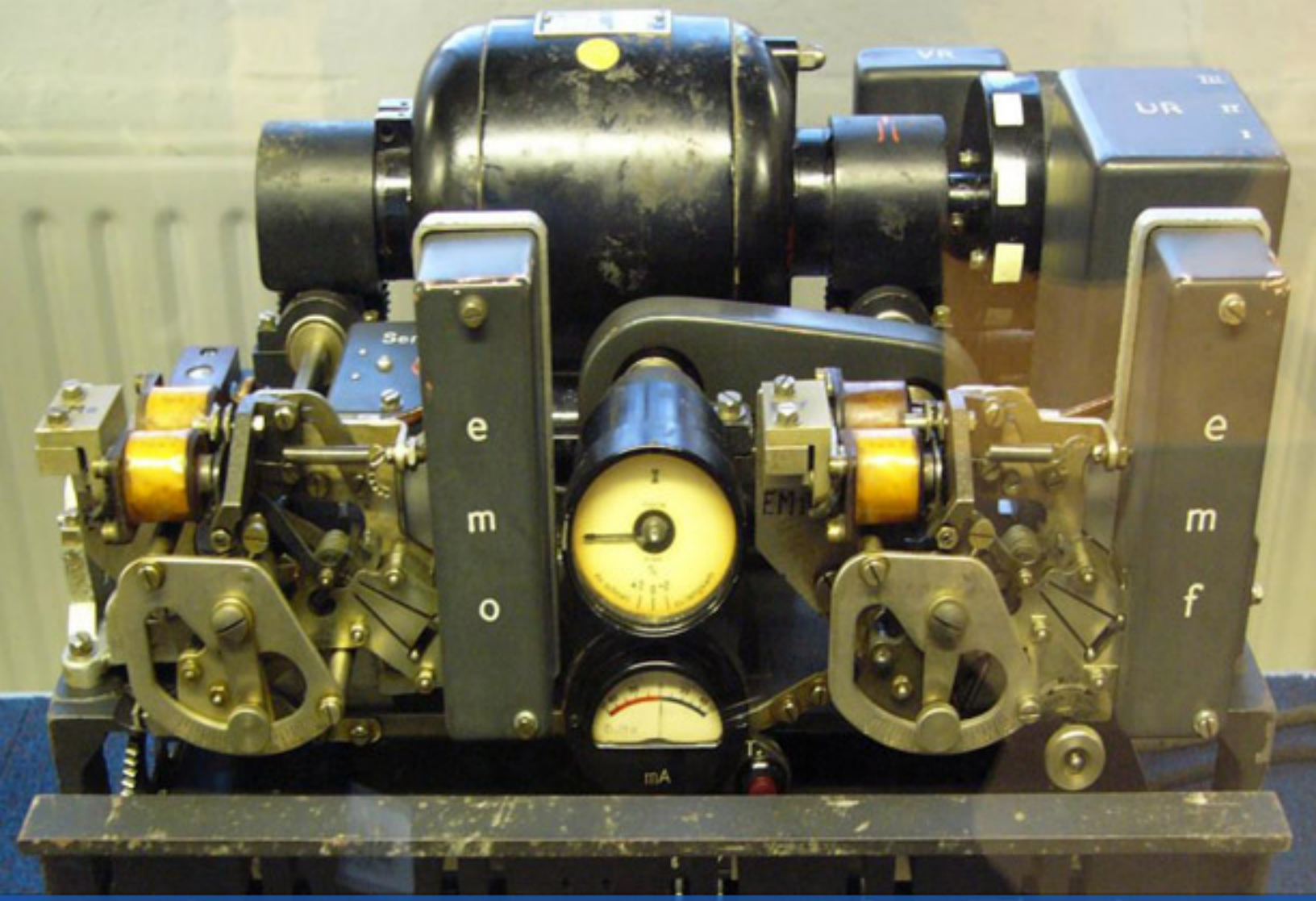


Handbook of Cryptographic Engineering

Amina Pedigo

First Edition, 2012

ISBN 978-81-323-4148-2

Published by:
White Word Publications
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Cryptographic Engineering

Cryptographic engineering is the discipline of using cryptography to solve human problems. Cryptography is typically applied when trying to ensure data confidentiality, to authenticate people or devices, or to verify data integrity in risky environments.

Cryptographic engineering is a complicated, multidisciplinary field. It encompasses mathematics (algebra, finite groups, rings, and fields), electrical engineering(hardware design, ASIC, FPGAs) and computer science (algorithms, complexity theory, software design, embedded systems). In order to practice state-of-the-art cryptographic design, mathematicians, computer scientists, and electrical engineers need to collaborate.

Below are the main topics that are specifically related to cryptographic engineering:

Cryptographic implementations

* Hardware architectures for public-key and secret-key cryptographic algorithms
* Cryptographic processors and co-processors
* Hardware accelerators for security protocols (security processors, network processors, etc.)
* True and pseudorandom number generators
* Physically unclonable functions (PUFs)
* Efficient software implementations of cryptography for embedded processors

Attacks against implementations and countermeasures against these attacks

* Side channel attacks and countermeasures
* Fault attacks and countermeasures
* Hardware tamper resistance
* Hardware trojans

Tools and methodologies

* Computer aided cryptographic engineering
* Verification methods and tools for secure design
* Metrics for the security of embedded systems

* Secure programming techniques

Applications

* Cryptography in wireless applications (mobile phone, WLANs, analysis of standards, etc.)
* Cryptography for pervasive computing (RFID, sensor networks, smart devices, etc.)
* FPGA design security
* Hardware IP protection and anti-counterfeiting
* Reconfigurable hardware for cryptography
* Smart card processors, systems and applications
* Security in commercial consumer applications (pay-TV, automotive, domotics, etc.)
* Secure storage devices (memories, disks, etc.)
* Technologies and hardware for content protection
* Trusted computing platforms

Interactions between cryptographic theory and implementation issues

* New and emerging cryptographic algorithms and protocols targeting embedded devices
* Non-classical cryptographic technologies
* Special-purpose hardware for cryptanalysis
* Formal methods for secure hardware

Major Issues

In modern practice, cryptographic engineering is deployed in crypto systems. Like most engineering design, these are wholly human creations. Most crypto systems are computer software, either embedded in firmware or running as ordinary executable files under an operating system. In some system designs, the cryptography runs under manual direction, in others, it is run automatically, often in the background. Like other software design, and unlike most other engineering, there are few external constraints.

Active opposition

In other engineering design, a successful design or implementation of one, is one which 'works'. Thus, an aircraft which actually flies without crashing due to some aerodynamic design blunder is a successful design. How successful is important, of course, and depends on how well it meets intended performance criteria. Continuing with the aircraft example, several World War I fighter aircraft designs only barely flew, while others flew well (at least one design flew well, but its wings broke off with some regularity) though with insufficient agility (turning, climbing, ..., rates) or insufficient stability (too frequent inescapable spins and so on) to be useful or survivable. To a considerable extent, good agility in aircraft is inversely related to inadequate stability, so fighter aircraft designs are, in this respect, inevitable compromises. The same considerations have continued in more recent times, as for instance the necessity for computer 'fly-by-wire' control in some fighters with great agility.

Cryptographic designs also have performance goals (eg, unbreakability of encryption), but must perform in a more complex, and more complexly hostile, environment than merely high (but not too low) in the Earth's atmosphere under war conditions.

Some aspects of the conditions under which crypto designs must work (to be successful and so worth bothering with) have been long recognized. Sensible cipher designers (of which there were fewer than their users would have wanted) attempted to find ways to prevent frequency analysis success, starting, it must be assumed, almost immediately after that cryptanalytic technique was first used. The most effective way to defeat frequency analysis attacks was the polyalphabetic substitution cipher, invented by Alberti about 1465. For the next several hundred years, other designers also tried to evade frequency analysis, usually poorly, demonstrating that few had a clear understanding of the problem. What is probably the best known (and likely the widest used) of those attempts is the (misnamed) Vigenère cipher which is a partial implementation of Alberti's idea. Edgar Allan Poe famously, and rashly, boasted that no cipher could defeat his cryptanalytic talents (essentially frequency analysis); that he was almost entirely correct about the ciphertexts submitted to him suggests a low level of cryptographic awareness some 400 (!) years after Alberti. As this history suggests, an important part of crypto engineering is understanding the techniques the Opposition may have available.

In addition, it has been explicitly realized since the mid-19th century that the Opposition must be credited with certain kinds of knowledge, lest one's design efforts address too little. Kerckhoffs' Law -- "The security of a cipher must reside entirely in the key", and the equivalent, and somewhat less obscure, Shannon's Maxim -- "The enemy knows the system", put it more or less clearly. A crypto design must achieve its goals (eg, confidentiality, or message integrity), not only despite active intelligent Opposition, but in spite of uncomfortably well informed Opposition.

Inherent zero-defect requirement

Many failures in cryptographic engineering are catastrophic. That is, success in breaking one message leads to reading all messages. Most cryptographic algorithms and protocols make certain assumptions (random key or nonce choices, for example), and when those assumptions are violated, all security is lost.

Examples: Netscape random bug found at UC Berkeley, Microsoft's PPTP protocol implementation problems found by Schneier.

Invisibility of most failure modes

Success in cryptographic engineering is unclear at best. Not crashing is a quite prominent *sine qua non* in aircraft design. Not allowing the Opposition access (to protected message traffic, for instance) is the design goal, but it is far less obvious when this goal has been achieved than in other engineering. Essentially no Opponents will ever make their access to message content public, and so neither designers nor implementors nor users of crypto

systems will ever learn from them that their design is insecure. It is certainly irrational to count on Opponents as a quality control resource.

One tempting measure of security is 'I can't figure out how to break it, so I will assume Opponents will not be able to do so either'. This may be true, but there is no way to actually know your Opponents have the same limitations you do. In a modern environment, in which messages travel over public networks, it is not even possible to detect eavesdropping, much less to prevent it. Accordingly, most message traffic must be presumed to be entirely in an Opponent's possession.

Known cryptographic failures fall into several classes. Future failures may also, or may find new categories. Examples include:

Design errors:

- cryptographic protocol errors
- user operational procedure errors
- algorithm implementation errors
- associated system failures

User errors:

- misunderstanding of correct operations
- arbitrary user actions

Implementation errors:

- programming errors (bugs)
- precision arithmetic errors
- random data errors
- software library routine errors

Environment errors:

- operating system insecurities with effects on cryptographic software (eg, keys retained in swap file data)
- operating system insecurities with regard to plaintext access
- operating system vulnerabilities (viruses, Trojan horses, etc)

The effect of most of these will not be apparent to end users, generally not to the computer system's administrators, and often not even to the cryptographic system's designers. For instance, a buffer overflow vulnerability in an obligatory operating system component may not have been present in version 5.1 (used during crypto system testing), but appear only at version 5.3, available only after release of the crypto system. Or that particular vulnerability may have been removed in all operating system releases later than version 5.3, but the crytographic system is being used in this case with version 5.1.

Chapter 2

Steganography

Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. The word *steganography* is of Greek origin and means "concealed writing" from the Greek words *steganos* (στεγανός) meaning "covered or protected", and *graphein* (γράφειν) meaning "to write". The first recorded use of the term was in 1499 by Johannes Trithemius in his *Steganographia*, a treatise on cryptography and steganography disguised as a book on magic. Generally, messages will appear to be something else: images, articles, shopping lists, or some other *covertext* and, classically, the hidden message may be in invisible ink between the visible lines of a private letter.

The advantage of steganography, over cryptography alone, is that messages do not attract attention to themselves. Plainly visible encrypted messages—no matter how unbreakable—will arouse suspicion, and may in themselves be incriminating in countries where encryption is illegal. Therefore, whereas cryptography protects the contents of a message, steganography can be said to protect both messages and communicating parties.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. As a simple example, a sender might start with an innocuous image file and adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it.

Ancient steganography

The first recorded uses of steganography can be traced back to 440 BC when Herodotus mentions two examples of steganography in *The Histories of Herodotus*. Demaratus sent a warning about a forthcoming attack to Greece by writing it directly on the wooden backing of a wax tablet before applying its beeswax surface. Wax tablets were in common use then as reusable writing surfaces, sometimes used for shorthand. Another

ancient example is that of Histiaeus, who shaved the head of his most trusted slave and tattooed a message on it. After his hair had grown the message was hidden. The purpose was to instigate a revolt against the Persians.

Steganographic techniques

Physical steganography

Steganart example. Within this picture, the letter positions of a hidden message are represented by increasing numbers (1 to 20), and a letter value is given by its intersection position in the grid. For instance, the first letter of the hidden message is at the intersection of 1 and 4. So, after a few tries, the first letter of the message seems to be the 14th letter of the alphabet; the last one (number 20) is the 5th letter of the alphabet.

Steganography has been widely used, including in recent historical times and the present day. Possible permutations are endless and known examples include:

- Hidden messages within wax tablets — in ancient Greece, people wrote messages on the wood, then covered it with wax upon which an innocent covering message was written.
- Hidden messages on messenger's body — also used in ancient Greece. Herodotus tells the story of a message tattooed on a slave's shaved head, hidden by the growth of his hair, and exposed by shaving his head again. The message allegedly carried a warning to Greece about Persian invasion plans. This method has obvious drawbacks, such as delayed transmission while waiting for the slave's hair to grow, and the restrictions on the number and size of messages that can be encoded on one person's scalp.
- During World War II, the French Resistance sent some messages written on the backs of couriers using invisible ink.
- Hidden messages on paper written in secret inks, under other messages or on the blank parts of other messages.
- Messages written in Morse code on knitting yarn and then knitted into a piece of clothing worn by a courier.
- Messages written on envelopes in the area covered by postage stamps.
- During and after World War II, espionage agents used photographically produced microdots to send information back and forth. Microdots were typically minute, approximately less than the size of the period produced by a typewriter. World War II microdots needed to be embedded in the paper and covered with an adhesive, such as collodion. This was reflective and thus detectable by viewing against glancing light. Alternative techniques included inserting microdots into slits cut into the edge of post cards.
- During World War II, a spy for Japan in New York City, Velvalee Dickinson, sent information to accommodation addresses in neutral South America. She was a dealer in dolls, and her letters discussed how many of this or that doll to ship. The stegotext was the doll orders, while the concealed "plaintext" was itself encoded and gave information about ship movements, etc. Her case became somewhat famous and she became known as the Doll Woman.
- Cold War counter-propaganda. In 1968, crew members of the USS *Pueblo* intelligence ship held as prisoners by North Korea, communicated in sign language during staged photo opportunities, informing the United States they were not defectors, but rather were being held captive by the North Koreans. In other photos presented to the U.S., crew members gave "the finger" to the unsuspecting North Koreans, in an attempt to discredit photos that showed them smiling and comfortable.

Digital steganography

Modern steganography entered the world in 1985 with the advent of the personal computer being applied to classical steganography problems. Development following that was slow, but has since taken off, going by the number of "stego" programs available:

Over 800 digital steganography applications have been identified by the Steganography Analysis and Research Center. Digital steganography techniques include:

- Concealing messages within the lowest bits of noisy images or sound files.
- Concealing data within encrypted data or within random data. The data to be concealed is first encrypted before being used to overwrite part of a much larger block of encrypted data or a block of random data (an unbreakable cipher like the one-time pad generates ciphertexts that look perfectly random if you don't have the private key).
- Chaffing and winnowing.
- Mimic functions convert one file to have the statistical profile of another. This can thwart statistical methods that help brute-force attacks identify the right solution in a ciphertext-only attack.
- Concealed messages in tampered executable files, exploiting redundancy in the targeted instruction set.
- Pictures embedded in video material (optionally played at slower or faster speed).
- Injecting imperceptible delays to packets sent over the network from the keyboard. Delays in keypresses in some applications (telnet or remote desktop software) can mean a delay in packets, and the delays in the packets can be used to encode data.
- Changing the order of elements in a set.
- Content-Aware Steganography hides information in the semantics a human user assigns to a datagram. These systems offer security against a non-human adversary/warden.
- Blog-Steganography. Messages are fractionalized and the (encrypted) pieces are added as comments of orphaned web-logs (or pin boards on social network platforms). In this case the selection of blogs is the symmetric key that sender and recipient are using; the carrier of the hidden message is the whole blogosphere.
- Modifying the echo of a sound file (Echo Steganography).
- Secure Steganography for Audio Signals.

Network steganography

All information hiding techniques that may be used to exchange steganograms in telecommunication networks can be classified under the general term of network steganography. This nomenclature was originally introduced by Krzysztof Szczypiorski in 2003. Contrary to the typical steganographic methods which utilize digital media (images, audio and video files) as a cover for hidden data, network steganography utilizes communication protocols' control elements and their basic intrinsic functionality. As a result, such methods are harder to detect and eliminate.

Typical network steganography methods involve modification of the properties of a single network protocol. Such modification can be applied to the PDU (Protocol Data Unit), to the time relations between the exchanged PDUs, or both (hybrid methods).

Moreover, it is feasible to utilize the relation between two or more different network protocols to enable secret communication. These applications fall under the term inter-protocol steganography.

Network steganography covers a broad spectrum of techniques, which include, among others:

- Steganophony - the concealment of messages in Voice-over-IP conversations, e.g. the employment of delayed or corrupted packets that would normally be ignored by the receiver (this method is called LACK - Lost Audio Packets Steganography), or, alternatively, hiding information in unused header fields.
- WLAN Steganography – the utilization of methods that may be exercised to transmit steganograms in Wireless Local Area Networks. A practical example of WLAN Steganography is the HICCUPS system (Hidden Communication System for Corrupted Networks)

Printed steganography

Digital steganography output may be in the form of printed documents. A message, the *plaintext*, may be first encrypted by traditional means, producing a *ciphertext*. Then, an innocuous *covertext* is modified in some way so as to contain the ciphertext, resulting in the *stegotext*. For example, the letter size, spacing, typeface, or other characteristics of a covertext can be manipulated to carry the hidden message. Only a recipient who knows the technique used can recover the message and then decrypt it. Francis Bacon developed Bacon's cipher as such a technique.

The ciphertext produced by most digital steganography methods, however, is not printable. Traditional digital methods rely on perturbing noise in the channel file to hide the message, as such, the channel file must be transmitted to the recipient with no additional noise from the transmission. Printing introduces much noise in the ciphertext, generally rendering the message unrecoverable. There are techniques that address this limitation, one notable example is ASCII Art Steganography.

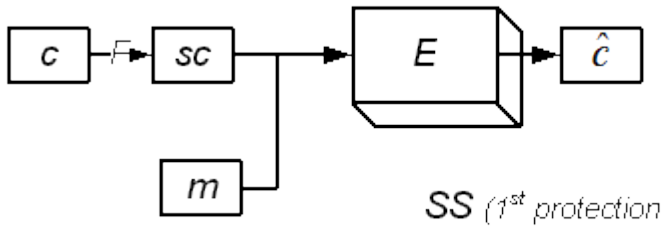
Text steganography

Steganography can be applied to different types of media including text, audio, image and video etc. However, text steganography is considered to be the most difficult kind of steganography due to lack of redundancy in text as compared to image or audio but still has smaller memory occupation and simpler communication. The method that could be used for text steganography is data compression. Data compression encodes information in one representation into another representation. The new representation of data is smaller in size. One of the possible schemes to achieve data compression is Huffman coding. Huffman coding assigns smaller length codewords to more frequently occurring source symbols and longer length codewords to less frequently occurring source symbols.

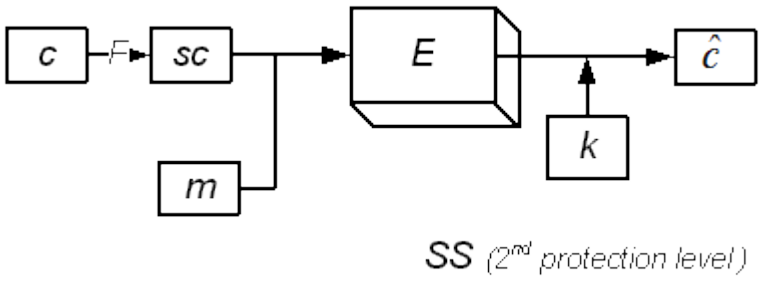
Steganography using Sudoku Puzzle

This is the art of concealing data in an image using Sudoku which is used like a key to hide the data within an image. Steganography using sudoku puzzles has as many keys as there are possible solutions of a Sudoku puzzle, which is 6.71×10^{21}. This is equivalent to around 70 bits, making it much stronger than the DES method which uses a 56 bit key.

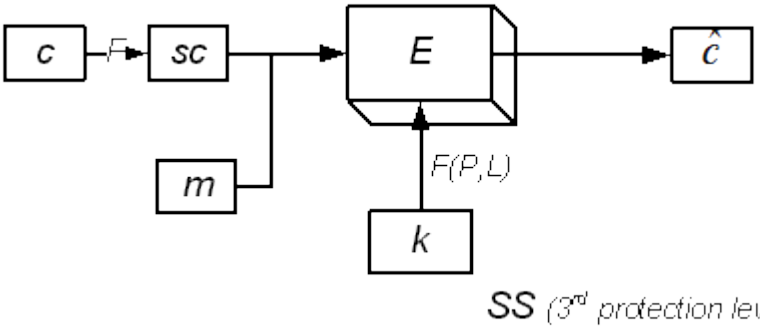
Data Embedding Security Schemes



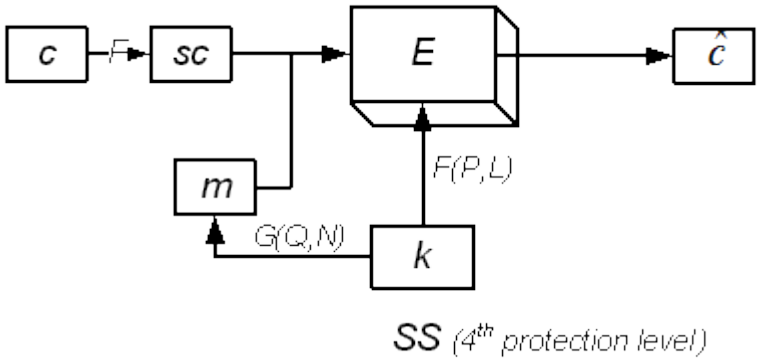
Steganographic System - The First Protection Level Scheme



Steganographic System - The Second Protection Level Scheme



Steganographic System - The Third Protection Level Scheme



Steganographic System - The Fourth Protection Level Scheme

The choice of embedding algorithm in the most cases is driven by the results of the steganographic channel robustness analysis . One of the areas that improve steganographic robustness is usage of a key scheme for embedding messages. Various key steganographic schemes have various levels of protection. Key scheme term means a procedure of how to use key steganographic system based on the extent of its use. However, when the steganographic robustness is increased a bandwidth of the whole embedding system is decreased. Therefore the task of a scheme selection for achieving the optimal values of the steganographic system is not trivial.

Embedding messages in steganographic system can be carried out without use of a key or with use of a key. To improve steganographic robustness key can be used as a verification option. It can make an impact on the distribution of bits of a message within a container, as well as an impact on the procedure of forming a sequence of embedded bits of a message.

The first level of protection is determined only by the choice of embedding algorithm. This may be the least significant bits modification algorithm, or algorithms for modifying the frequency or spatial-temporal characteristics of the container. The first level of protection is presented in any steganographic channel. Steganographic system in this case can be represented as shown at *The First Protection Level Scheme* figure. There following notations are used: c - is a container file; F - steganographic channel space (frequency or/and amplitude container part, that is available for steganographic modification and message signal transmission); SC - steganographic system; m - message to be embedded; E - embedding method; $\hat{c}$ - modified container file.

The second protection level of the steganographic system, as well as all levels of protection of the higher orders, is characterized by the use of Key (password) via steganographic modification. An example of a simple key scheme, which provides a second level of protection, is to write the unmodified or modified password in the top or bottom of the message; or the distribution of the password sign on the entire length of the steganographic channel. Such key schemes do not affect the distribution of messages through the container and do not use a message preprocessing according to the defined key. This kind of steganographic systems are used in such tasks as, for instance, adding a

digital signature for proof of copyright. Data embedding performance is not changed in comparison with the fastest approach of the first protection level usage.

Steganographic data channels that use key schemes based distribution of a message through the container and or preprocessing of an embedded message for data hiding are more secure. When the third protection level key scheme is used it affects the distribution of a message through the container – distribution function of a message within a container; P – minimum number of container samples that are needed to embed one message sample; L – step of a message distribution within a container). Accordingly, the performance of container processing will be lower than in the case of the first and the second key schemes. Taking into account that $P \geq L$, the simplest representation of the $F(P, L)$ function could be as following:

$$F(P, L) = cycle*L + step*P,$$

where *cycle* is a number of the current L section and *step* is a number of the embedded message sample.

The difference between the fourth protection level scheme and the third one is that in steganographic system there are two distribution functions of a message within a container are used. The first is responsible for a message samples selection according to some function $G(Q, N)$, and the second function $F(P, L)$ is responsible for position selection in a container for message sample hiding. Here Q – the size of message block to be inserted; N – the size (in bits) of one sample of the message file.

Based on the above discussion it is possible to define a classification table of key steganographic schemes:

Key Steganographic Schemes Classification

Steganographic system protection level	Steganographic algorithm usage	Key (password) usage	Key influence on a message signal bits distribution per container	Key influence on a message signal bits selection and distribution per container
1	+	-	-	-
2	+	+	-	-
3	+	+	+	-
4	+	+	+	+

Additional terminology

In general, terminology analogous to (and consistent with) more conventional radio and communications technology is used; however, a brief description of some terms which

show up in software specifically, and are easily confused, is appropriate. These are most relevant to digital steganographic systems.

The *payload* is the data to be covertly communicated. The *carrier* is the signal, stream, or data file into which the payload is hidden; which differs from the "*channel*" (typically used to refer to the type of input, such as "a JPEG image"). The resulting signal, stream, or data file which has the payload encoded into it is sometimes referred to as the *package*, *stego file*, or *covert message*. The percentage of bytes, samples, or other signal elements which are modified to encode the payload is referred to as the *encoding density* and is typically expressed as a number between 0 and 1.

In a set of files, those files considered likely to contain a payload are called *suspects*. If the *suspect* was identified through some type of statistical analysis, it might be referred to as a *candidate*.

Countermeasures and detection

Detection of physical steganography requires careful physical examination, including the use of magnification, developer chemicals and ultraviolet light. It is a time-consuming process with obvious resource implications, even in countries where large numbers of people are employed to spy on their fellow nationals. However, it is feasible to screen mail of certain suspected individuals or institutions, such as prisons or prisoner-of-war (POW) camps. During World War II, a technology used to ease monitoring of POW mail was specially treated paper that would reveal invisible ink. An article in the June 24, 1948 issue of *Paper Trade Journal* by the Technical Director of the United States Government Printing Office, Morris S. Kantrowitz, describes in general terms the development of this paper, three prototypes of which were named *Sensicoat*, *Anilith*, and *Coatalith* paper. These were for the manufacture of post cards and stationery to be given to German prisoners of war in the US and Canada. If POWs tried to write a hidden message the special paper would render it visible. At least two US patents were granted related to this technology, one to Mr. Kantrowitz, No. 2,515,232, "Water-Detecting paper and Water-Detecting Coating Composition Therefor", patented July 18, 1950, and an earlier one, "Moisture-Sensitive Paper and the Manufacture Thereof", No. 2,445,586, patented July 20, 1948. A similar strategy is to issue prisoners with writing paper ruled with a water-soluble ink that "runs" when in contact with a water-based invisible ink.

In computing, detection of steganographically encoded packages is called steganalysis. The simplest method to detect modified files, however, is to compare them to known originals. For example, to detect information being moved through the graphics on a website, an analyst can maintain known-clean copies of these materials and compare them against the current contents of the site. The differences, assuming the carrier is the same, will compose the payload. In general, using extremely high compression rate makes steganography difficult, but not impossible. While compression errors provide a hiding place for data, high compression reduces the amount of data available to hide the payload in, raising the encoding density and facilitating easier detection (in the extreme case, even by casual observation).

Applications

Usage in modern printers

Steganography is used by some modern printers, including HP and Xerox brand color laser printers. Tiny yellow dots are added to each page. The dots are barely visible and contain encoded printer serial numbers, as well as date and time stamps.

Example from modern practice

The larger the cover message is (in data content terms—number of bits) relative to the hidden message, the easier it is to hide the latter. For this reason, digital pictures (which contain large amounts of data) are used to hide messages on the Internet and on other communication media. It is not clear how commonly this is actually done. For example: a 24-bit bitmap will have 8 bits representing each of the three color values (red, green, and blue) at each pixel. If we consider just the blue there will be 2^8 different values of blue. The difference between 11111111 and 11111110 in the value for blue intensity is likely to be undetectable by the human eye. Therefore, the least significant bit can be used (more or less undetectably) for something else other than color information. If we do it with the green and the red as well we can get one letter of ASCII text for every three pixels.

Stated somewhat more formally, the objective for making steganographic encoding difficult to detect is to ensure that the changes to the carrier (the original signal) due to the injection of the payload (the signal to covertly embed) are visually (and ideally, statistically) negligible; that is to say, the changes are indistinguishable from the noise floor of the carrier. Any medium can be a carrier, but media with a large amount of redundant or compressible information are better suited.

From an information theoretical point of view, this means that the channel must have more capacity than the "surface" signal requires; that is, there must be redundancy. For a digital image, this may be noise from the imaging element; for digital audio, it may be noise from recording techniques or amplification equipment. In general, electronics that digitize an analog signal suffer from several noise sources such as thermal noise, flicker noise, and shot noise. This noise provides enough variation in the captured digital information that it can be exploited as a noise cover for hidden data. In addition, lossy compression schemes (such as JPEG) always introduce some error into the decompressed data; it is possible to exploit this for steganographic use as well.

Steganography can be used for digital watermarking, where a message (being simply an identifier) is hidden in an image so that its source can be tracked or verified (for example, Coded Anti-Piracy), or even just to identify an image (as in the EURion constellation)

Alleged use by terrorists

When one considers that messages could be encrypted steganographically in e-mail messages, particularly e-mail spam, the notion of junk e-mail takes on a whole new light. Coupled with the "chaffing and winnowing" technique, a sender could get messages out and cover their tracks all at once.

An example showing how terrorists may use forum avatars to send hidden messages.

Rumors about terrorists using steganography started first in the daily newspaper *USA Today* on February 5, 2001 in two articles titled "Terrorist instructions hidden online" and "Terror groups hide behind Web encryption". In July the same year, an article was titled even more precisely: "Militants wire Web with links to jihad". A citation from the article: "*Lately, al-Qaeda operatives have been sending hundreds of encrypted messages that have been hidden in files on digital photographs on the auction site eBay.com*". Other media worldwide cited these rumors many times, especially after the terrorist attack of 9/11, without ever showing proof. The Italian newspaper *Corriere della Sera* reported that an Al Qaeda cell which had been captured at the Via Quaranta mosque in Milan had pornographic images on their computers, and that these images had been used to hide secret messages (although no other Italian paper ever covered the story). The *USA Today* articles were written by veteran foreign correspondent Jack Kelley, who in 2004 was fired after allegations emerged that he had fabricated stories and sources.

In October 2001, the *New York Times* published an article claiming that al-Qaeda had used steganography to encode messages into images, and then transported these via e-mail and possibly via USENET to prepare and execute the September 11, 2001 terrorist attack. The Federal Plan for Cyber Security and Information Assurance Research and Development, published in April 2006 makes the following statements:

- "...immediate concerns also include the use of cyberspace for covert communications, particularly by terrorists but also by foreign intelligence services; espionage against sensitive but poorly defended data in government and industry systems; subversion by insiders, including vendors and contractors; criminal activity, primarily involving fraud and theft of financial or identity information, by hackers and organized crime groups..." (p. 9–10)

- "International interest in R&D for steganography technologies and their commercialization and application has exploded in recent years. These technologies pose a potential threat to national security. Because steganography secretly embeds additional, and nearly undetectable, information content in digital

products, the potential for covert dissemination of malicious software, mobile code, or information is great." (p. 41–42)

- "The threat posed by steganography has been documented in numerous intelligence reports." (p. 42)

Moreover, an online "terrorist training manual", the "Technical Mujahid, a Training Manual for Jihadis" contained a section entitled "Covert Communications and Hiding Secrets Inside Images."

By early 2002, a Cranfield University MSc thesis developed the first practical implementation of an online real-time Counter Terrorist Steganography Search Engine. This was designed to detect the most likely image steganography in transit and thereby provide UK Ministry of Defence Intelligence Staff a realistic approach to "narrowing the field", suggesting that interception capacity was never the difficulty but rather prioritising the target media.

Despite this, *there are no known instances of terrorists using computer steganography*. Al Qaeda's use of steganography is somewhat simpler: In 2008 a British man, Rangzieb Ahmed, was alleged to have a contact book with Al-Qaeda telephone numbers, written in invisible ink. He was convicted of terrorism.

Alleged use by intelligence services

In 2010, the Federal Bureau of Investigation revealed that the Russian foreign intelligence service uses customized steganography software for embedding encrypted text messages inside image files for certain communications with "illegal agents" (agents under non-diplomatic cover) stationed abroad.

Chapter 3

Autokey Cypher

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

A tabula recta for use with an autokey cypher

An **autokey cypher** (also known as the **autoclave cypher**) is a cypher which incorporates the message (the plaintext) into the key. There are two forms of autokey cypher: *key autokey* and *text autokey* cyphers. A key-autokey cypher uses previous

members of the keystream to determine the next element in the keystream. A text-autokey uses the previous message text to determine the next element in the keystream.

In modern cryptography, self-synchronizing stream cyphers are autokey cyphers.

History

The first autokey cypher was invented by Girolamo Cardano, and contained a fatal defect. Like many autokey cyphers it used the plaintext to encrypt itself; however, since there was no additional key, it is no easier for the intended recipient to read the message than anyone else who knows that the cypher is being used. A number of attempts were made by other cryptographers to produce a system that was neither trivial to break nor too difficult for the intended recipient to decypher. Eventually one was invented in 1564 by Giovan Battista Bellaso using a "reciprocal table" with five alphabets of his invention and another form was described in 1586 by Blaise de Vigenère with a similar reciprocal table of ten alphabets. This system is now known as the Vigenère cypher.

One popular form of autokey starts with a tabula recta, a square with 26 copies of the alphabet, the first line starting with 'A', the next line starting with 'B', etc., like the one above. In order to encrypt a plaintext, one locates the row with the first letter to be encrypted, and the column with the first letter of the key. The letter where the line and column cross is the cyphertext letter.

Giovan Battista Bellaso used the first letter of each word as a primer to start his text autokey. Blaise de Vigenère used as a primer an agreed-upon single letter of the alphabet.

The autokey cypher as used by the members of the American Cryptogram Association differs in the way the key is generated. It starts with a relatively short keyword, and appends the message to it. So if the keyword is "QUEENLY", and the message is "ATTACK AT DAWN", the key would be "QUEENLYATTACKATDAWN".

```
Plaintext:  ATTACK AT DAWN...
Key:        QUEENL YA TTACK AT DAWN....
cyphertext: QNXEPV YT WTWP...
```

The cyphertext message would therefore be "QNXEPVYTWTWP".

Cryptanalysis

Using an example message "meet at the fountain" encrypted with the keyword "KILT":

```
plaintext:  MEETATTHEFOUNTAIN (unknown)
key:        KILTMEETATTHEFOUN (unknown)
cyphertext: WMPMMXXAEYHBRYOCA (known)
```

We try common words, bigrams, trigrams etc. in all possible positions in the key. For example, "THE":

```
cyphertext: WMP MMX XAE YHB RYO CA
key:        THE THE THE THE THE ..
plaintext:  DFL TFT ETA FAX YRK ..

cyphertext: W MPM MXX AEY HBR YOC A
key:        . THE THE THE THE THE .
plaintext:  . TII TQT HXU OUN FHY .

cyphertext: WM PMM XXA EYH BRY OCA
key:        .. THE THE THE THE THE
plaintext:  .. WFI EQW LRD IKU VVW
```

We sort the plaintext fragments in order of likelihood:

```
unlikely <------------------> promising
EQW DFL TFT ... ... ... ... ETA OUN FAX
```

We know that a correct plaintext fragment will also appear in the key, shifted right by the length of the keyword. Similarly our guessed key fragment ("THE") will also appear in the plaintext shifted left. So by guessing keyword lengths (probably between 3 and 12) we can reveal more plaintext and key.

Trying this with "OUN" (possibly after wasting some time with the others):

```
shift by 4:
cyphertext: WMPMMXXAEYHBRYOCA
key:        ......ETA.THE.OUN
plaintext:  ......THE.OUN.AIN

by 5:
cyphertext: WMPMMXXAEYHBRYOCA
key:        .....EQW..THE..OU
plaintext:  .....THE..OUN..OG

by 6:
cyphertext: WMPMMXXAEYHBRYOCA
key:        ....TQT...THE...O
plaintext:  ....THE...OUN...M
```

We see that a shift of 4 looks good (both of the others have unlikely Qs), so we shift the revealed "ETA" back by 4 into the plaintext:

```
cyphertext: WMPMMXXAEYHBRYOCA
key:        ..LTM.ETA.THE.OUN
plaintext:  ..ETA.THE.OUN.AIN
```

We have a lot to work with now. The keyword is probably 4 characters long ("..LT"), and we have some of the message:

```
M.ETA.THE.OUN.AIN
```

Because our plaintext guesses have an effect on the key 4 characters to the left, we get feedback on correct/incorrect guesses, so we can quickly fill in the gaps:

```
MEETATTHEFOUNTAIN
```

The ease of cryptanalysis is thanks to the feedback from the relationship between plaintext and key. A 3-character guess reveals 6 more characters, which then reveal further characters, creating a cascade effect, allowing us to rule out incorrect guesses quickly.

Autokey in modern cyphers

Modern autokey cyphers use very different encryption methods, but they follow the same approach of using either key bytes or plaintext bytes to generate more key bytes. Most modern stream cyphers are based on pseudorandom number generators: the key is used to initialize the generator, and either key bytes or plaintext bytes are fed back into the generator to produce more bytes.

Some stream cyphers are said to be "self-synchronizing", because the next key byte usually depends only on the previous N bytes of the message. If a byte in the message is lost or corrupted, therefore, the key-stream will also be corrupted--but only until N bytes have been processed. At that point the keystream goes back to normal, and the rest of the message will decrypt correctly.

Chapter 4

Blind Signature

In cryptography, a **blind signature**, as introduced by David Chaum, is a form of digital signature in which the content of a message is disguised (blinded) before it is signed. The resulting blind signature can be publicly verified against the original, unblinded message in the manner of a regular digital signature. Blind signatures are typically employed in privacy-related protocols where the signer and message author are different parties. Examples include cryptographic election systems and digital cash schemes.

An often-used analogy to the cryptographic blind signature is the physical act of enclosing a message in a special write-through-capable envelope, which is then sealed and signed by a signing agent. Thus, the signer does not view the message content, but a third party can later verify the signature and know that the signature is valid within the limitations of the underlying signature scheme.

Blind signatures can also be used to provide *unlinkability*, which prevents the signer from linking the blinded message it signs to a later un-blinded version that it may be called upon to verify. In this case, the signer's response is first "un-blinded" prior to verification in such a way that the signature remains valid for the un-blinded message. This can be useful in schemes where anonymity is required.

Blind signature schemes can be implemented using a number of common public key signing schemes, for instance RSA and DSA. To perform such a signature, the message is first "blinded", typically by combining it in some way with a random "blinding factor". The blinded message is passed to a signer, who then signs it using a standard signing algorithm. The resulting message, along with the blinding factor, can be later verified against the signer's public key. In some blind signature schemes, such as RSA, it is even possible to remove the blinding factor from the signature before it is verified. In these schemes, the final output (message/signature) of the blind signature scheme is identical to that of the normal signing protocol.

Uses

Blind signature schemes see a great deal of use in applications where sender privacy is important. This includes various "digital cash" schemes and voting protocols.

For example, the integrity of some electronic voting system may require that each ballot be certified by an election authority before it can be accepted for counting; this allows the authority to check the credentials of the voter to ensure that they are allowed to vote, and that they are not submitting more than one ballot. Simultaneously, it is important that this authority not learn the voter's selections. An unlinkable blind signature provides this guarantee, as the authority will not see the contents of any ballot it signs, and will be unable to link the blinded ballots it signs back to the un-blinded ballots it receives for count

Blind signature schemes

Blind signature schemes exist for many public key signing protocols. Some examples are provided below. In each example, the message to be signed is contained in the value *m*. *m* is considered to be some legitimate input to the signature function. As an analogy, consider that Alice has a letter which should be signed by an authority (say Bob), but Alice does not want to reveal the content of the letter to Bob. She can place the letter in an envelope lined with carbon paper and send it to Bob. Bob will sign the outside of the carbon envelope without opening it and then send it back to Alice. Alice can then open it to find the letter signed by Bob, but without Bob having seen its contents.

More formally a blind signature scheme is a cryptographic protocol that involves two parties, a user Alice that wants to obtain signatures on her messages, and a signer Bob that is in possession of his secret signing key. At the end of the protocol Alice obtains a signature on *m* without Bob learning anything about the message. This intuition of not learning anything is hard to capture in mathematical terms. The usual approach is to show that for every (adversarial) signer, there exists a simulator that can output the same information as the signer. This is similar to the way zero-knowledge is defined in zero-knowledge proof systems.

Blind RSA signatures

One of the simplest blind signature schemes is based on RSA signing. A traditional RSA signature is computed by raising the message *m* to the secret exponent *d* modulo the public modulus *N*. The blind version uses a random value *r*, such that *r* is relatively prime to *N* (i.e. $gcd(r, N) = 1$). *r* is raised to the public exponent *e* modulo *N*, and the resulting value $r^e \bmod N$ is used as a blinding factor. The author of the message computes the product of the message and blinding factor, i.e.

$$m' \equiv mr^e \pmod{N}$$

and sends the resulting value *m*' to the signing authority. Because *r* is a random value and the mapping $r \mapsto r^e \bmod N$ is a permutation it follows that $r^e \bmod N$ is random too. This implies that *m*' does not leak any information about *m*. The signing authority then calculates the blinded signature *s′* as:

$$s' \equiv (m')^d \pmod{N}.$$

s' is sent back to the author of the message, who can then remove the blinding factor to reveal s, the valid RSA signature of m:

$$s \equiv s' \cdot r^{-1} \pmod{N}$$

This works because RSA keys satisfy the equation $r^{ed} \equiv r \pmod{N}$ and thus

$$s \equiv s' \cdot r^{-1} \equiv (m')^d r^{-1} \equiv m^d r^{ed} r^{-1} \equiv m^d r r^{-1} \equiv m^d \pmod{N},$$

hence s is indeed the signature of m.

Dangers of blind signing

RSA is subject to the RSA blinding attack through which it is possible to be tricked into decrypting a message by blind signing another message. Since the signing process is equivalent to decrypting with the signers secret key, an attacker can provide a blinded version of a message m encrypted with the signers public key, m' for them to sign. The encrypted message would usually be some secret information which the attacker observed being sent encrypted under the signers public key which the attacker wants to learn. When the attacker unblinds the signed version they will have the clear text:

$$\begin{aligned} m'' &= m' r^e \pmod{n} \\ &= (m^e \pmod{n} \cdot r^e) \pmod{n} \\ &= (mr)^e \pmod{n} \end{aligned}$$

where m' is the encrypted version of the message. When the message is signed, the cleartext m is easily extracted:

$$\begin{aligned} s' &= m''^d \pmod{n} \\ &= ((mr)^e \pmod{n})^d \pmod{n} \\ &= (mr)^{ed} \pmod{n} \\ &= m \cdot r \pmod{n}, \text{ since } ed \equiv 1 \pmod{\phi(n)} \end{aligned}$$

Note that $\varphi(n)$ refers to Euler's totient function. The message is now easily obtained.

$$m = s' \cdot r^{-1} \pmod{n}$$

This attack works because in this blind signature scheme the signer signs the message directly. By contrast, in an unblinded signature scheme the signer would typically use a padding scheme (e.g. by instead signing the result of a Cryptographic hash function applied to the message, instead of signing the message itself), however since the signer

does not know the actual message, any padding scheme would produce an incorrect value when unblinded. Due to this multiplicative property of RSA, the same key should never be used for both encryption and signing purposes.

Chapter 5

Backdoor (Computing) and Array Controller Based Encryption

Backdoor (computing)

A **backdoor** in a computer system (or cryptosystem or algorithm) is a method of bypassing normal authentication, securing remote access to a computer, obtaining access to plaintext, and so on, while attempting to remain undetected. The backdoor may take the form of an installed program (e.g., Back Orifice) or may subvert the system through a rootkit.

Overview

The threat of backdoors surfaced when multiuser and networked operating systems became widely adopted. Petersen and Turn discussed computer subversion in a paper published in the proceedings of the 1967 AFIPS Conference. They noted a class of active infiltration attacks that use "trapdoor" entry points into the system to bypass security facilities and permit direct access to data. The use of the word *trapdoor* here clearly coincides with more recent definitions of a backdoor. However, since the advent of public key cryptography the term *trapdoor* has acquired a different meaning. More generally, such security breaches were discussed at length in a RAND Corporation task force report published under ARPA sponsorship by J.P. Anderson and D.J. Edwards in 1970.

A backdoor in a login system might take the form of a hard coded user and password combination which gives access to the system. A famous example of this sort of backdoor was as a plot device in the 1983 film *WarGames*, in which the architect of the "WOPR" computer system had inserted a hardcoded password (his dead son's name) which gave the user access to the system, and to undocumented parts of the system (in particular, a video game–like simulation mode and direct interaction with the artificial intelligence).

An attempt to plant a backdoor in the Linux kernel, exposed in November 2003, showed how subtle such a code change can be. In this case, a two-line change appeared to be a

typographical error, but actually gave the caller to the `sys_wait4` function root access to the system.

Although the number of backdoors in systems using proprietary software (software whose source code is not publicly available) is not widely credited, they are nevertheless frequently exposed. Programmers have even succeeded in secretly installing large amounts of benign code as Easter eggs in programs, although such cases may involve official forbearance, if not actual permission.

It is also possible to create a backdoor without modifying the source code of a program, or even modifying it after compilation. This can be done by rewriting the compiler so that it recognizes code during compilation that triggers inclusion of a backdoor in the compiled output. When the compromised compiler finds such code, it compiles it as normal, but also inserts a backdoor (perhaps a password recognition routine). So, when the user provides that input, he gains access to some (likely undocumented) aspect of program operation. This attack was first outlined by Ken Thompson in his famous paper *Reflections on Trusting Trust* (see below).

Many computer worms, such as Sobig and Mydoom (and the covert Skynet), install a backdoor on the affected computer (generally a PC on broadband running insecure versions of Microsoft Windows and Microsoft Outlook). Such backdoors appear to be installed so that spammers can send junk e-mail from the infected machines. Others, such as the Sony/BMG rootkit distributed silently on millions of music CDs through late 2005, are intended as DRM measures — and, in that case, as data gathering agents, since both surreptitious programs they installed routinely contacted central servers.

A traditional backdoor is a symmetric backdoor: anyone that finds the backdoor can in turn use it. The notion of an asymmetric backdoor was introduced by Adam Young and Moti Yung in the *Proceedings of Advances in Cryptology: Crypto '96*. An asymmetric backdoor can only be used by the attacker who plants it, even if the full implementation of the backdoor becomes public (e.g., via publishing, being discovered and disclosed by reverse engineering, etc.). Also, it is computationally intractable to detect the presence of an asymmetric backdoor under black-box queries. This class of attacks have been termed kleptography; they can be carried out in software, hardware (for example, smartcards), or a combination of the two. The theory of asymmetric backdoors is part of a larger field now called cryptovirology.

There exists an experimental asymmetric backdoor in RSA key generation. This OpenSSL RSA backdoor was designed by Young and Yung, utilizes a twisted pair of elliptic curves, and has been made available.

Reflections on Trusting Trust

Ken Thompson's *Reflections on Trusting Trust* was the first major paper to describe black box backdoor issues, and points out that trust is relative. It described a very clever backdoor mechanism based upon the fact that people only review source (human-written)

code, and not compiled machine code. A program called a compiler is used to create the second from the first, and the compiler is usually trusted to do an honest job.

Thompson's paper described a modified version of the Unix C compiler that would:

- Put an invisible backdoor in the Unix login command when it noticed that the login program was being compiled, and as a twist
- Also add this feature undetectably to future compiler versions upon *their* compilation as well.

Because the compiler itself was a compiled program, users would be extremely unlikely to notice the machine code instructions that performed these tasks. (Because of the second task, the compiler's source code would appear "clean".) What's worse, in Thompson's proof of concept implementation, the subverted compiler also subverted the analysis program (the disassembler), so that anyone who examined the binaries in the usual way would not actually see the real code that was running, but something else instead. This version was, officially, never released into the wild. It is believed, however, that a version was distributed to BBN and at least one use of the backdoor was recorded.

This attack was recently (August 2009) discovered by Sophos labs: The W32/Induc-A virus infected the program compiler for Delphi, a Windows programming language. The virus introduced its own code to the compilation of new Delphi programs, allowing it to infect and propagate to many systems, without the knowledge of the software programmer. An attack that propagates by building its own Trojan Horse can be especially hard to discover. It is believed that the Induc-A virus had been propagating for at least a year before it was discovered.

Once a system has been compromised with a backdoor or Trojan horse, such as the *Trusting Trust* compiler, it is very hard for the "rightful" user to regain control of the system. However, several practical weaknesses in the *Trusting Trust* scheme have been suggested. For example, a sufficiently motivated user could painstakingly review the machine code of the untrusted compiler before using it. As mentioned above, there are ways to hide the trojan horse, such as subverting the disassembler; but there are ways to counter that defense, too, such as writing your own disassembler from scratch, so the infected compiler won't recognize it. However, such proposals are generally impractical. If a user had a serious concern that the compiler was compromised, they would be better off avoiding using it altogether rather than reviewing the binary in detail using only tools that have been verified to be untainted. A user that did not have serious concerns that the compiler was compromised could not be practically expected to undertake the vast amount of work required.

David A. Wheeler has proposed a counter to this attack using an approach he calls "diverse double-compiling", which uses techniques adapted from compiler bootstrapping. This involves re-compiling the source of the compiler through another independently-written and generated "trusted" compiler, and then using the binary generated from this to recompile the original compiler again, and then comparing the binary generated from this

second compilation with that generated from using the original compiler to recompile itself directly.

Array controller based encryption

Within a storage network, encryption of data may occur at different hardware levels. **Array controller based encryption** describes the encryption of data occurring at the disk array controller before being sent to the disk drives. Here we will provide an overview of different implementation techniques to array controller based encryption.

Possible Points of Encryption in SAN

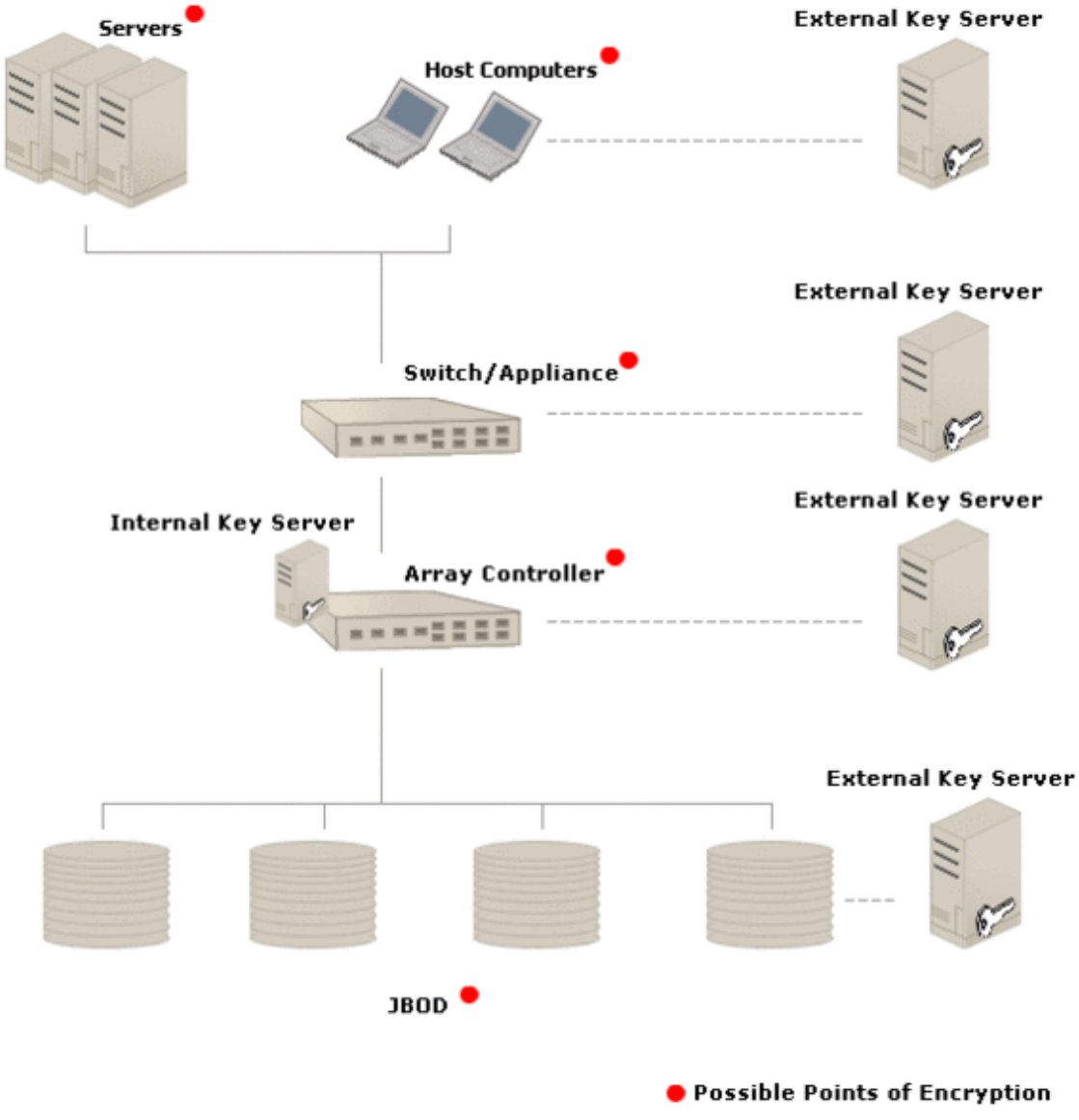


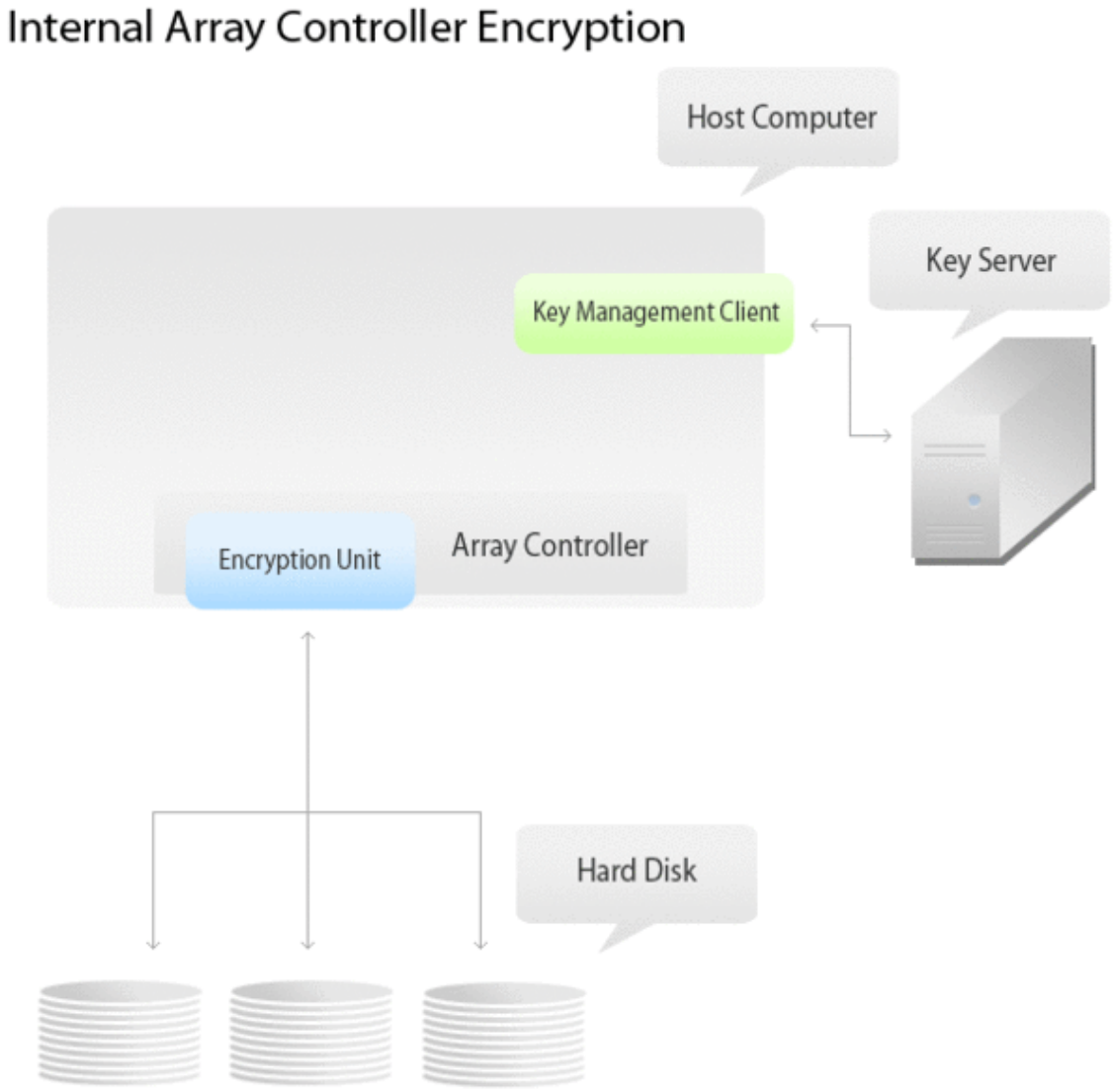
Diagram showing the possible points of encryption within a storage network

The encryption of data can take place in many points in a storage network. The point of encryption may occur on the host computer, in the SAN infrastructure, the array controller or on each of the hard disks as shown on the diagram above. Each point of encryption has different merits and costs. Within the diagram, the key server components are also shown for each configuration of encryption. Designers of SANs and SAN components must take into consideration factors such as performance, deployment complexity, key server interoperability, strength of security, and cost when choosing where to implement encryption. But since the array controller is a natural central point of all data therefore encryption at this level is inherent and also reduces deployment complexity.

Array Controller Based Encryption

With different configurations of a hardware or software array controller, there are different types of solutions for this type of encryption. Each of these solutions can be built into existing infrastructures by replacing or upgrading certain components. Basic components include an encryption key server, key management client, and commonly an encryption unit which are all implemented into a storage network.

Internal Array Controller Encryption



Encryption implementation in an internal array controller architecture

For an internal array controller configuration, the array controller is generally a PCI bus card situated inside the host computer. As shown in the diagram, the PCI array controller would contain an encryption unit where plaintext data is encrypted into ciphertext. This separate encryption unit is utilized to prevent and minimize performance reduction and maintain data throughput. Furthermore, the Key Management Client will generally be an additional service within the host computer applications where it will authenticate all keys retrieved from the Key Server. A major disadvantage to this type of implementation would be that encryption components are required to be integrated within each host computer and therefore is redundant on large networks with many host devices.

External Array Controller Encryption

In the case of an external array controller setup, the array controller would be an independent hardware module connected to the network. Within the hardware array controller would be an Encryption unit for data encryption as well as a Key Management Client for authentication. Generally, there are few hardware array controllers to many host devices and storage disks. Therefore it reduces deployment complexity to implement into fewer hardware components. Moreover, the lifecycle of an array controller is generally much longer than host computers and storage disks, therefore the encryption implementation will not need to be reimplemented as often as if encryption was done at another point in the storage network.

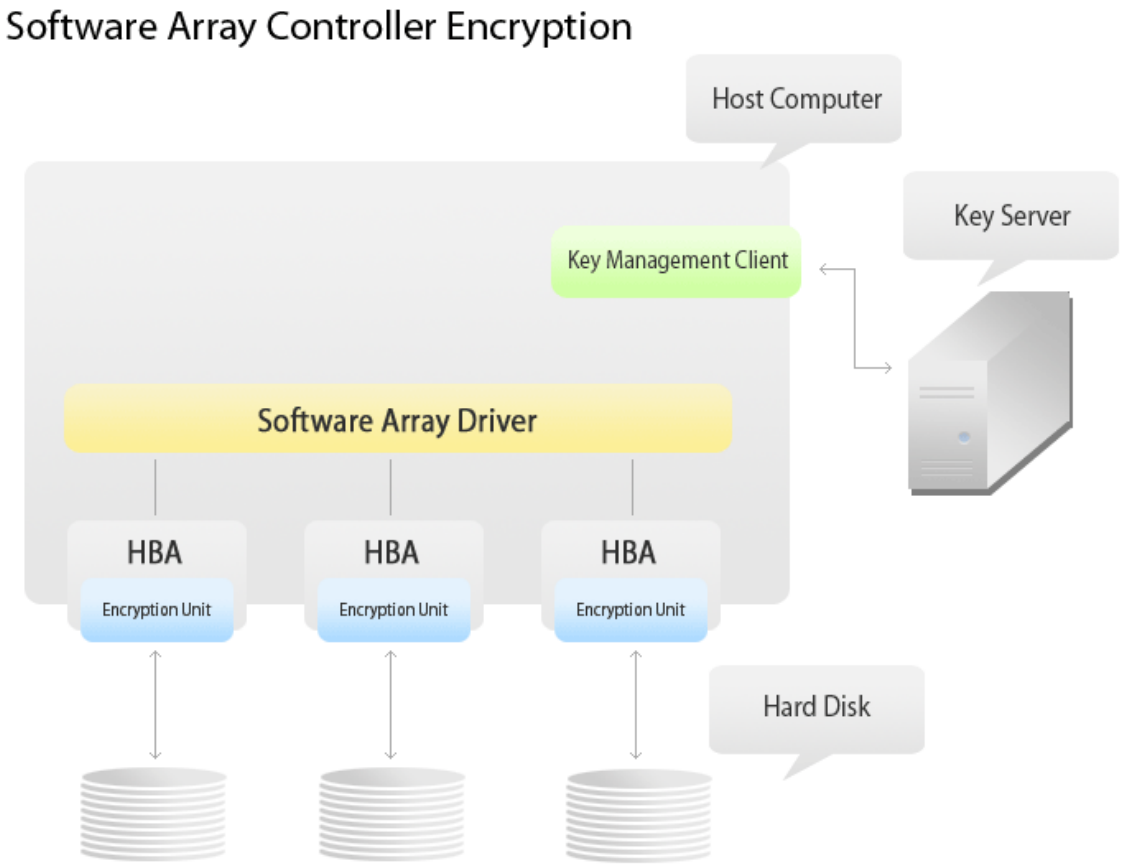
Encryption at the Front-End or Back-End Side Array Controller

In an external array controller, the encryption unit can either be placed either on the front-end side or the back-end side of the array controller. There are different advantages and disadvantages in placing the encryption unit either on the front-end side or the back-end side:

	Advantages	Disadvantages
Front-End Side	All data is first encrypted before it moves along the array controller, therefore data is encrypted before sending it through the replication link and or stored in internal array controller cache.	Since data is encrypted before it moves along the array controller, data de-duplication and data compression cannot be done when sending data through replication link. Therefore huge costs can be incurred when sending huge amounts of data through the replication link.
Back-End Side	Since all data is encrypted before leaving the array controller, data de-duplication and data compression can be done and therefore may save costs since only compressed and unique data is sent through the replication link.	Sensitive data may be compromised when sending through the replication link as well as cached data in the array controller compromised.

The placement of the encryption unit may highly impact the secureness of your controller based encryption implementation. Therefore this issue must be taken account for when designing your implementation to mitigate all security risks.

Software Array Controller Encryption



Encryption implementation in an software array controller architecture

For the software array controller encryption, a software array controller driver directs data into individual host bus adapters. In the diagram on the right, there are multiple host bus adapters with hardware encryption units used for better performance requirements. In contrast, this type of encryption can be implemented with only 1 host bus adapter connected to a network of multiple hard drives and would still function. Performance will definitely be reduced since there will only be one encryption unit processing data. Key management will be done much like the internal array controller encryption mentioned before with the Key Management Client implemented as a service within the Host Computer.

Chapter 6

Cryptovirology and Ciphertext

Cryptovirology

Cryptovirology is a field that studies how to use cryptography to design powerful malicious software. The field was born with the observation that public-key cryptography can be used to break the symmetry between what an antivirus analyst sees regarding a virus and what the virus writer sees. The former only sees a public key whereas the latter sees a public key and corresponding private key. The first attack that was identified in the field is called "cryptoviral extortion". In this attack a virus, worm, or trojan hybrid encrypts the victim's files and the user must pay the malware author to receive the needed session key (which is encrypted under the author's public key that is contained in the malware) if the user does not have backups and needs the files back.

The field also encompasses covert attacks in which the attacker secretly steals private information such as private keys. An example of the latter type of attack are asymmetric backdoors. An **asymmetric backdoor** is a backdoor (e.g., in a cryptosystem) that can be used only by the attacker, even after it is found. This contrasts with the traditional backdoor that is symmetric, i.e., anyone that finds it can use it. Kleptography, a subfield of cryptovirology, is concerned with the study of asymmetric back doors in key generation algorithms, digital signature algorithms, key exchanges, and so on.

General information

Cryptovirology was born in academia. However, practitioners have recently expanded the scope of the field to include the analysis of cryptographic algorithms used by malware writers, attacks on these algorithms using automated methods (such as X-raying) and analysis of viruses' and packers' encryptors. Also included is the study of cryptography-based techniques (such as "delayed code") developed by malware writers to hamper malware analysis.

A "questionable encryption scheme", which was introduced by Young and Yung, is an attack tool in cryptovirology. Informally speaking, a questionable encryption scheme is a public key cryptosystem (3-tuple of algorithms) with two supplementary algorithms, forming a 5-tuple of algorithms. It includes a deliberately bogus yet carefully designed key pair generation algorithm that produces a "fake" public key. The corresponding

private key (witness of non-encryption) cannot be used to decipher data "encrypted" using the fake public key. By supplying the key pair to an efficient verification predicate (the 5th algorithm in the 5-tuple) it is proven whether the public key is real or fake. When the public key is fake, it follows that no one can decipher data "enciphered" using the fake public key. A questionable encryption scheme has the property that real public keys are computationally indistinguishable from fake public keys when the private key is not available. The private key forms a poly-sized witness of decipherability or indecipherability, whichever may be the case.

An application of a questionable encryption scheme is a trojan that gathers plaintext from the host, "encrypts" it using the trojan's own public key (which may be real or fake), and then exfiltrates the resulting "ciphertext". In this attack it is thoroughly intractable to prove that data theft has occurred. This holds even when all core dumps of the trojan and all the information that it broadcasts is entered into evidence. An analyst that jumps to the conclusion that the trojan "encrypts" data risks being proven wrong by the malware author (e.g., anonymously).

When the public key is fake, the attacker gets no plaintext from the trojan. So what's the use? A spoofing attack is possible in which some trojans are released that use real public keys and steal data and some trojans are released that use fake public keys and do not steal data. Many months after the trojans are discovered and analyzed, the attacker anonymously posts the witnesses of non-encryption for the fake public keys. This proves that those trojans never in fact exfiltrated data. This casts doubt on the true nature of future strains of malware that contain such "public keys", since the keys could be real or fake. This attack implies a fundamental limitation on proving data theft.

There are many other attacks in the field of cryptovirology that are not mentioned here.

Examples of viruses with cryptography and ransom capabilities

While viruses in the wild have used cryptography in the past, the only purpose of such usage of cryptography was to avoid detection by antivirus software. For example, the tremor virus used polymorphism as a defensive technique in an attempt to avoid detection by anti-virus software. Though cryptography does assist in such cases to enhance the longevity of a virus, the capabilities of cryptography are not used in the payload. The One-half virus was amongst the first viruses known to have encrypted affected files. However, the One_half virus was not ransomware, that is it did not demand any ransom for decrypting the files that it has encrypted. It also did not use public key cryptography. An example of a virus that informs the owner of the infected machine to pay a ransom is the virus nicknamed Tro_Ransom.A . This virus asks the owner of the infected machine to send $10.99 to a given account through Western Union.
Virus.Win32.Gpcode.ag is a classic cryptovirus . This virus partially uses a version of 660-bit RSA and encrypts files with many different extensions. It instructs the owner of the machine to email a given mail ID if the owner desires the decryptor. If contacted by email, the user will be asked to pay a certain amount as ransom in return for the decryptor.

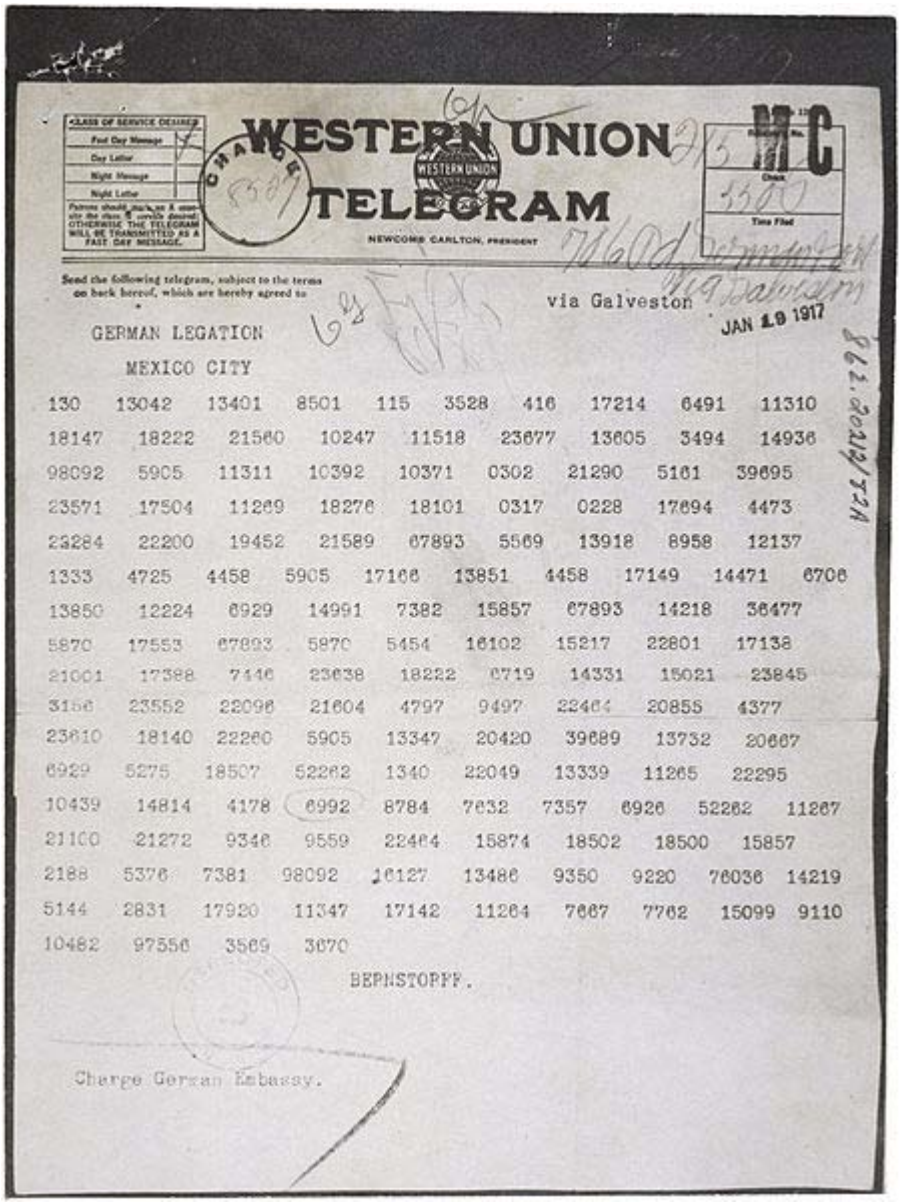
Creation of cryptoviruses

To successfully write a cryptovirus, a thorough knowledge of the various cryptographic primitives such as random number generators, proper recommended cipher text chaining modes etc are necessary. Wrong choices can lead to poor cryptographic strength. So, usage of preexisting routines would be ideal. Microsoft's Cryptographic API (CAPI), is a possible tool for the same. It has been demonstrated that using just 8 different calls to this API, a cryptovirus can satisfy all its encryption needs.

Other uses of cryptography enabled malware

Apart from cryptoviral extortion, there are other potential uses of cryptoviruses. They are used in deniable password snatching, used with cryptocounters, used with private information retrieval and used in secure communication between different instances of a distributed cryptovirus.

Ciphertext



WESTERN UNION TELEGRAM

NEWCOMB CARLTON, PRESIDENT

Send the following telegram, subject to the terms on back hereof, which are hereby agreed to

via Galveston

JAN 19 1917

GERMAN LEGATION

MEXICO CITY

130 13042 13401 8501 115 3528 416 17214 6491 11310
18147 18222 21560 10247 11518 23677 13605 3494 14936
98092 5905 11311 10392 10371 0302 21290 5161 39695
23571 17504 11269 18276 18101 0317 0228 17694 4473
23284 22200 19452 21589 07893 5569 13918 8958 12137
1333 4725 4458 5905 17166 13851 4458 17149 14471 6706
13850 12224 6929 14991 7382 15857 67893 14218 36477
5870 17553 67893 5870 5454 16102 15217 22801 17138
21001 17388 7446 23638 18222 6719 14331 15021 23845
3156 23552 22098 21604 4797 9497 22464 20855 4377
23610 18140 22260 5905 13347 20420 39689 13732 20667
6929 5275 18507 52262 1340 22049 13339 11265 22295
10439 14814 4178 6992 8784 7632 7357 6926 52262 11267
21100 21272 9346 9559 22464 15874 18502 18500 15857
2188 5376 7381 98092 16127 13486 9350 9220 76036 14219
5144 2831 17920 11347 17142 11264 7667 7762 15099 9110
10482 97556 3569 3070

BERNSTORFF.

Charge German Embassy.

The Zimmermann Telegram (as it was sent from Washington to Mexico) encrypted as ciphertext.

In cryptography, **ciphertext** (or **cyphertext**) is the result of encryption performed on plaintext using an algorithm, called a cipher. Ciphertext is also known as encrypted or encoded information because it contains a form of the original plaintext that is unreadable by a human or computer without the proper cipher to decrypt it. Decryption, the inverse of encryption, is the process of turning ciphertext into readable plaintext. Ciphertext is not to be confused with codetext because the latter is a result of a Code, not a cipher.

Symmetric key example

Let $m$be the plaintext message that Alice wants to secretly transmit to Bob and let $E_k$be the encryption cipher, where $k$is a secret key. Alice must first transform the plaintext into ciphertext, c, in order to securely send the message to Bob.

$$c = E_k(m)$$

Both Alice and Bob must know the choice of key, k, or else the ciphertext is useless. Once the message is encrypted as ciphertext, Alice can safely transmit it to Bob (assuming no one else knows the key). In order to read Alice's message, Bob must decrypt the ciphertext using ${E_k}^{-1}$which is known as the decryption cipher, D_k.

$$D_k(c) = D_k(E_k(m)) = m$$

Types of ciphers

The history of cryptography begins thousands of years ago and contains a variety of different types of encryption. Earlier algorithms were performed by hand and are substantially different from modern algorithms, which are generally executed by a machine.

Historical ciphers

Historical pen and paper ciphers used in the past are sometimes known as classical ciphers. They include:

- **Substitution cipher**: the units of plaintext are replaced with ciphertext (Caesar cipher and One-time pad)
- **Transposition cipher**: the ciphertext is a permutation of the plaintext (Rail fence cipher)
- **Polyalphabetic substitution cipher**: a substitution cipher using multiple substitution alphabets (Vigenère cipher and Enigma machine)
- **Permutation cipher**: a transposition cipher in which the key is a permutation

Historical ciphers are not generally not used as a standalone encryption solution because they are quite easy to crack. Many of the classical ciphers can be cracked using brute force or by analyzing only ciphertext with the exception of the one-time pad.

Modern ciphers

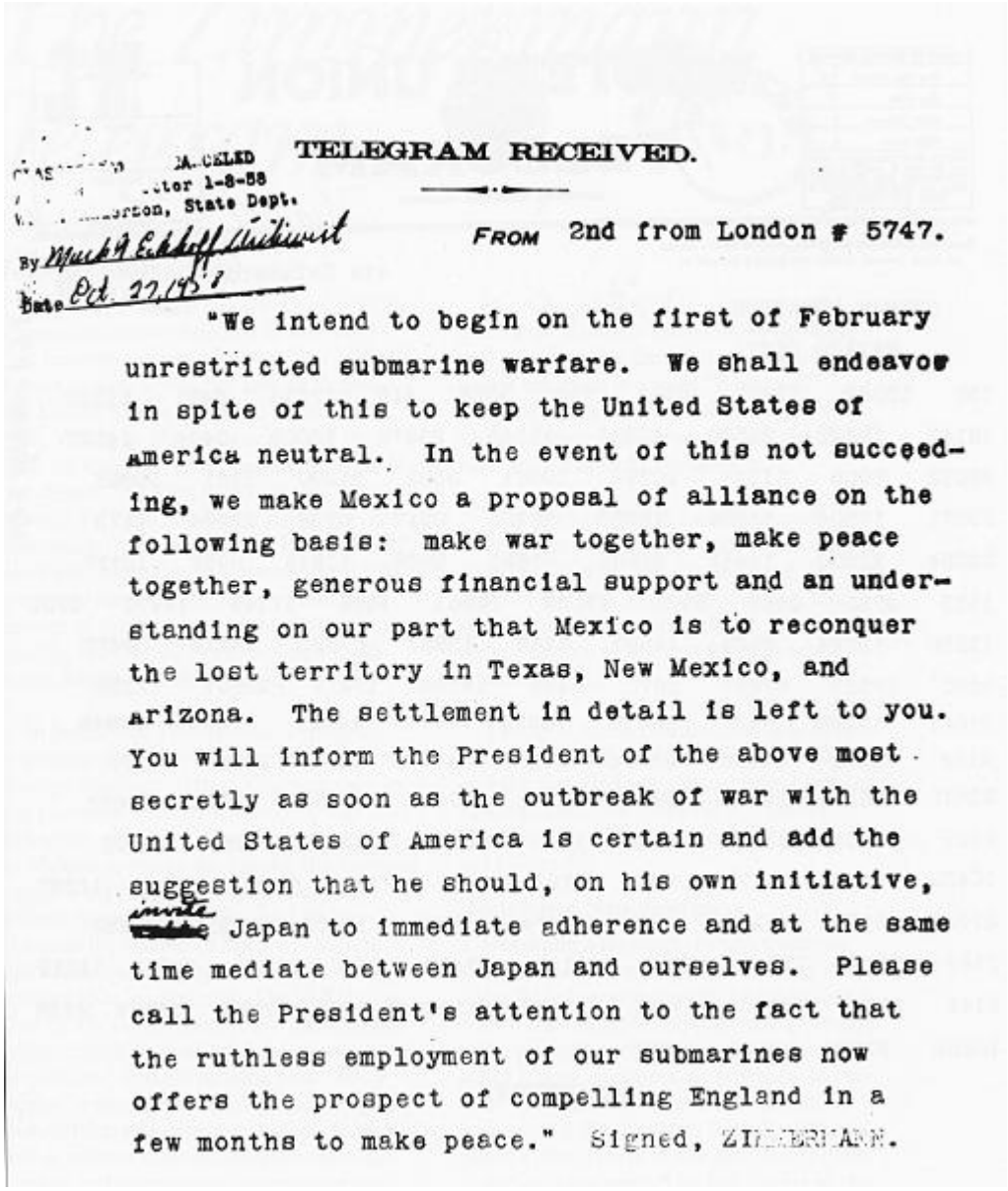
Modern ciphers are more secure than classical ciphers and are designed to withstand a wide range of attacks. An attacker should not be able to find the key used in a modern cipher, even if he knows any amount of plaintext and corresponding ciphertext. Modern encryption methods can be divided into the following categories:

- **Private-key cryptography** (symmetric key algorithm): the same key is used for encryption and decryption
- **Public-key cryptography** (asymmetric key algorithm): two different keys are used for encryption and decryption

In a symmetric key algorithm (e.g., DES and AES), the sender and receiver must have a shared key set up in advance and kept secret from all other parties; the sender uses this key for encryption, and the receiver uses the same key for decryption. In an asymmetric key algorithm (e.g., RSA), there are two separate keys: a *public key* is published and enables any sender to perform encryption, while a *private key* is kept secret by the receiver and enables only him to perform correct decryption.

Symmetric key ciphers can be divided into block ciphers and stream ciphers. Block ciphers operate on fixed-length groups of bits, called blocks, with an unvarying transformation. Stream ciphers encrypt plaintext digits one at a time on a continuous stream of data and he transformation of successive digits varies during the encryption process.

Cryptanalysis



TELEGRAM RECEIVED.

FROM 2nd from London # 5747.

"We intend to begin on the first of February unrestricted submarine warfare. We shall endeavor in spite of this to keep the United States of America neutral. In the event of this not succeeding, we make Mexico a proposal of alliance on the following basis: make war together, make peace together, generous financial support and an understanding on our part that Mexico is to reconquer the lost territory in Texas, New Mexico, and Arizona. The settlement in detail is left to you. You will inform the President of the above most secretly as soon as the outbreak of war with the United States of America is certain and add the suggestion that he should, on his own initiative, ~~invite~~ Japan to immediate adherence and at the same time mediate between Japan and ourselves. Please call the President's attention to the fact that the ruthless employment of our submarines now offers the prospect of compelling England in a few months to make peace." Signed, ZIMMERMANN.

The Zimmermann Telegram decrypted into plaintext (and translated into English).

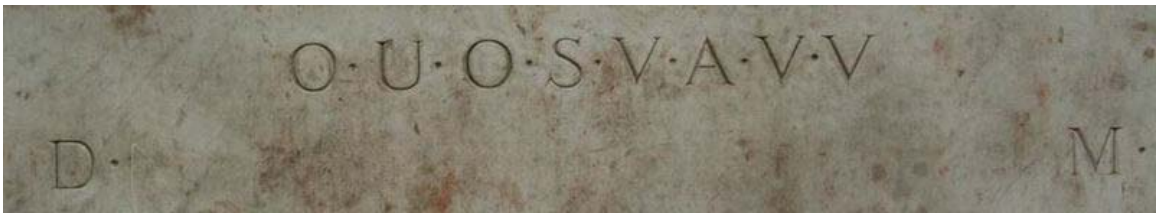
Cryptanalysis is the study of methods for obtaining the meaning of encrypted information, without access to the secret information that is normally required to do so. Typically, this involves knowing how the system works and finding a secret key. Cryptanalysis is also referred to as codebreaking or cracking the code. Ciphertext is generally the easiest part of a cryptosystem to obtain and therefore is a important part of cryptanalysis. Depending on what information is available and what type of cipher is being analyzed, crypanalysts can follow one or more attack models to crack a cipher.

Attack Models

- **Ciphertext-only**: the cryptanalyst has access only to a collection of ciphertexts or codetexts.
- **Known-plaintext**: the attacker has a set of ciphertexts to which he knows the corresponding plaintext.
- **Chosen-plaintext attack**: the attacker can obtain the ciphertexts corresponding to an arbitrary set of plaintexts of his own choosing.
    - **Batch chosen-plaintext attack**: where the cryptanalyst chooses all plaintexts before any of them are encrypted. This is often the meaning of an unqualified use of "chosen-plaintext attack".
    - **Adaptive chosen-plaintext attack**: where the cryptanalyst makes a series of interactive queries, choosing subsequent plaintexts based on the information from the previous encryptions.
- **Chosen-ciphertext attack**: the attacker can obtain the plaintexts corresponding to an arbitrary set of ciphertexts of his own choosing.
    - **Adaptive chosen-ciphertext attack**
    - **Indifferent chosen-ciphertext attack**
- **Related-key attack**: like a chosen-plaintext attack, except the attacker can obtain ciphertexts encrypted under two different keys. The keys are unknown, but the relationship between them is known; for example, two keys that differ in the one bit.

The ciphertext-only attack model is the weakest attack because it implies that the cryptanalyst has nothing but ciphertext. Modern ciphers rarely fail under this attack.

Famous ciphertexts



The Shugborough inscription, England

- The Babington Plot ciphers
- The Shugborough inscription
- The Zimmermann Telegram
- The Magic Words are Squeamish Ossifrage
- The cryptogram in "The Gold-Bug"
- Beale ciphers
- Kryptos
- Zodiac Killer ciphers

Chapter 7

Code (Cryptography)

In cryptography, a **code** is a method used to transform a message into an obscured form, preventing those who do not possess special information, or key, required to apply the transform from understanding what is actually transmitted. The usual method is to use a *codebook* with a list of common phrases or words matched with a *codeword*. Encoded messages are sometimes termed *codetext*, while the original message is usually referred to as plaintext.

Terms like *code* and *in code* are often used to refer to any form of encryption. However, there is an important distinction between codes and ciphers in technical work; it is, essentially, the scope of the transformation involved. Codes operate at the level of meaning; that is, words or phrases are converted into something else. Ciphers work at the level of individual letters, or small groups of letters, or even, in modern ciphers, with individual bits. While a code might transform "change" into "CVGDK" or "cocktail lounge", a cipher transforms elements below the semantic level, i.e., below the level of meaning. The "a" in "attack" might be converted to "Q", the first "t" to "f", the second "t" to "3", and so on. Ciphers are more convenient than codes in some situations, there being no need for a codebook, with its inherently limited number of valid messages, and the possibility of fast automatic operation on computers.

Codes were long believed to be more secure than ciphers, since (if the compiler of the codebook did a good job) there is no pattern of transformation which can be discovered, whereas ciphers use a consistent transformation, which can potentially be identified and reversed (except in the case of the one-time pad).

One- and two-part codes

Codes are defined by "codebooks" (physical or notional), which are dictionaries of codegroups listed with their corresponding plaintext. Codes originally had the codegroups assigned in 'plaintext order' for convenience of the code designed, or the encoder. For example, in a code using numeric code groups, a plaintext word starting with "a" would have a low-value group, while one starting with "z" would have a high-value group. The same codebook could be used to "encode" a plaintext message into a coded message or "codetext", and "decode" a codetext back into plaintext message.

However, such "one-part" codes had a certain predictability that made it easier for others to notice patterns and "crack" or "break" the message, revealing the plaintext, or part of it. In order to make life more difficult for codebreakers, codemakers designed codes with no predictable relationship between the codegroups and the ordering of the matching plaintext. In practice, this meant that two codebooks were now required, one to find codegroups for encoding, the other to look up codegroups to find plaintext for decoding. Students of foreign languages work much the same way; for, say, a Frenchman studying English, there is need of both an English-French and a French-English dictionary. Such "two-part" codes required more effort to develop, and twice as much effort to distribute (and discard safely when replaced), but they were harder to break.

One-time code

A **one-time code** is a prearranged word, phrase or symbol that is intended to be used only once to convey a simple message, often the signal to execute or abort some plan or confirm that it has succeeded or failed. One time codes are often designed to be included in what would appear to be an innocent conversation. Done properly they are almost impossible to detect, though a trained analyst monitoring the communications of someone who has already aroused suspicion might be able to recognize a comment like "Aunt Bertha has gone into labor" as having an ominous meaning. Famous example of one time codes include:

- "One if by land; two if by sea" in "Paul Revere's Ride" made famous in the poem by Henry Wadsworth Longfellow
- "Climb Mount Niitaka" - the signal to Japanese planes to begin the attack on Pearl Harbor
- During World War II the British Broadcasting Corporation's overseas service frequently included "personal messages" as part of its regular broadcast schedule. The seemingly nonsensical stream of messages read out by announcers were actually one time codes intended for SOE agents operating behind enemy lines. An example might be "The princess wears red shoes" or "Mimi's cat is asleep under the table". Each code message was read out twice. By such means, the French Resistance were instructed to start sabotaging rail and other transport links the night before D-day.
- "Over all of Spain, the sky is clear" was a signal (broadcast on radio) to start the nationalist military revolt in Spain on July 17, 1936.

Sometimes messages are not prearranged and rely on shared knowledge hopefully known only to the recipients. An example is the telegram sent to U.S. President Harry Truman, then in Potsdam to meet with Stalin, informing Truman of the first successful test of an atomic bomb.

"Operated on this morning. Diagnosis not yet complete but results seem satisfactory and already exceed expectations. Local press release necessary as interest extends great distance. Dr. Groves pleased. He returns tomorrow. I will keep you posted."

Idiot code

An *idiot code* is a code that is created by the parties using it. This type of communication is akin to the hand signals used by armies in the field.

Example: Any sentence where 'day' and 'night' are used means 'attack'. The location mentioned in the following sentence specifies the location to be attacked.

- *Plaintext:* Attack Gotham.
- *Codetext:* We walked day and night through the streets but couldn't find it! Tomorrow we'll head into Gotham.

An early use of the term appears to be by George Perrault, a character in the science fiction book *Friday* by Robert A. Heinlein:

The simplest sort [of code] and thereby impossible to break. The first ad told the person or persons concerned to carry out number seven or expect number seven or it said something about something designated as seven. This one says the same with respect to code item number ten. But the meaning of the numbers cannot be deduced through statistical analysis because the code can be changed long before a useful statistical universe can be reached. It's an idiot code... and an idiot code can never be broken if the user has the good sense not to go too often to the well.

Richard Miniter, author of *Losing Bin Laden: How Bill Clinton's Failures Unleashed Global Terror*, was quoted in an interview by UPI Technology News:

Another way terrorists use the Internet to communicate is through conventional message boards. They simply go to common public places online, chat rooms and the like, and post messages using what intelligence operatives call an "idiot code", said Miniter.

Terrorism expert Magnus Ranstorp said that the men who carried out the September 11, 2001, attacks on the United States used basic e-mail and what he calls "idiot code" to discuss their plans.

Cryptanalysis of codes

While solving a monoalphabetic substitution cipher is easy, solving even a simple code is difficult. Decrypting a coded message is a little like trying to translate a document written in a foreign language, with the task basically amounting to building up a "dictionary" of the codegroups and the plaintext words they represent.

One fingerhold on a simple code is the fact that some words are more common than others, such as "the" or "a" in English. In telegraphic messages, the codegroup for "STOP" (i.e., end of sentence or paragraph) is usually very common. This helps define the structure of the message in terms of sentences, if not their meaning, and this is cryptanalytically useful.

Further progress can be made against a code by collecting many codetexts encrypted with the same code and then using information from other sources

- spies,
- newspapers,
- diplomatic cocktail party chat,
- the location from where a message was sent,
- where it was being sent to (i.e., traffic analysis)
- the time the message was sent,
- events occurring before and after the message was sent
- the normal habits of the people sending the coded messages
- etc.

For example, a particular codegroup found almost exclusively in messages from a particular army and nowhere else might very well indicate the commander of that army. A codegroup that appears in messages preceding an attack on a particular location may very well stand for that location.

Of course, cribs can be an immediate giveaway to the definitions of codegroups. As codegroups are determined, they can gradually build up a critical mass, with more and more codegroups revealed from context and educated guesswork. One-part codes are more vulnerable to such educated guesswork than two-part codes, since if the codenumber "26839" of a one-part code is determined to stand for "bulldozer", then the lower codenumber "17598" will likely stand for a plaintext word that starts with "a" or "b". At least, for simple one part codes.

Various tricks can be used to "plant" or "sow" information into a coded message, for example by executing a raid at a particular time and location against an enemy, and then examining code messages sent after the raid. Coding errors are a particularly useful fingerhold into a code; people reliably make errors, sometimes disastrous ones. Of course, planting data and exploiting errors works against ciphers as well.

- The most obvious and, in principle at least, simplest way of cracking a code is to steal the codebook through bribery, burglary, or raiding parties — procedures sometimes glorified by the phrase "practical cryptography" — and this is a weakness for both codes and ciphers, though codebooks are generally larger and used longer than cipher keys. While a good code may be harder to break than a cipher, the need to write and distribute codebooks is seriously troublesome.

Constructing a new code is like building a new language and writing a dictionary for it; it was an especially big job before computers. If a code is compromised, the entire task must be done all over again, and that means a lot of work for both cryptographers and the code users. In practice, when codes were in widespread use, they were usually changed on a periodic basis to frustrate codebreakers, and to limit the useful life of stolen or copied codebooks.

Once codes have been created, codebook distribution is logistically clumsy, and increases chances the code will be compromised. Codes can be thought reasonably secure if they are only used by a few careful people, but if whole armies use the same codebook, security becomes much more difficult.

In contrast, the security of ciphers is generally dependent on protecting the cipher keys. Cipher keys can be stolen and people can betray them, but they are much easier to change and distribute.

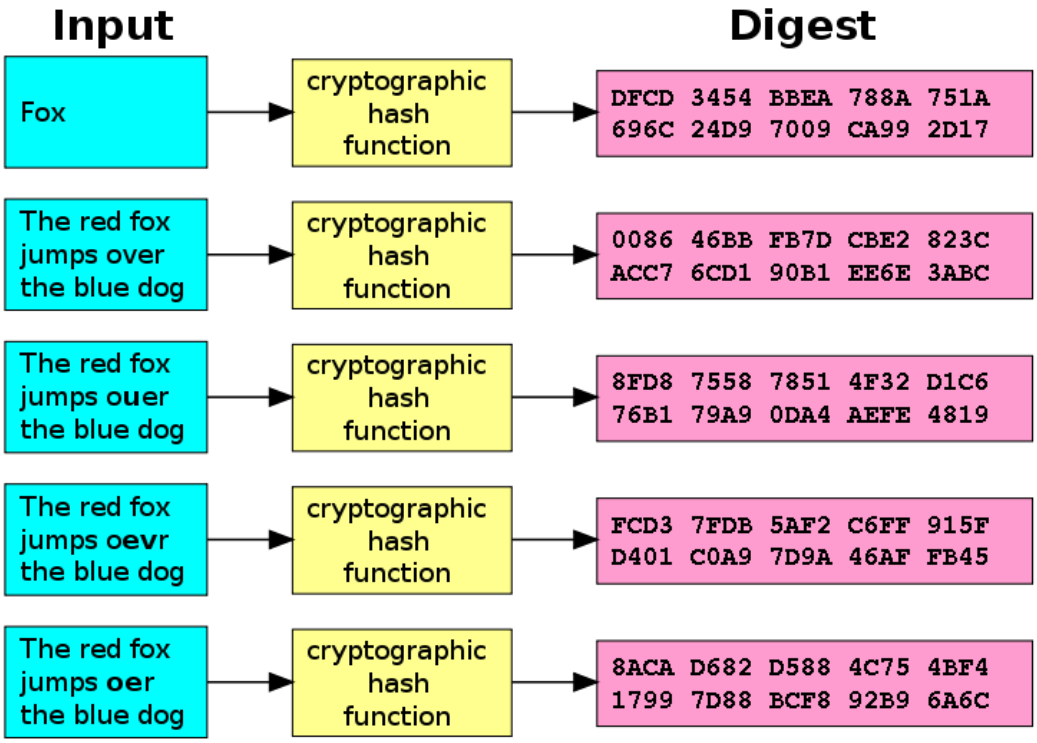
Superencipherment

In more recent practice, it became typical to encipher a message after first encoding it, so as to provide greater security by increasing the degree of difficulty for cryptanalysts. With a numerical code, this was commonly done with an "additive" - simply a long key number which was digit-by-digit added to the code groups, modulo 10. Unlike the codebooks, additives would be changed frequently. The famous Japanese Navy code, JN-25, was of this design, as were several of the (confusingly named) Royal Navy Cyphers used after WWI and into WWII.

One might wonder why a code would be used if it had to be enciphered to provide security. As well as providing security, a well designed code can also compress the message, and provide some degree of automatic error correction. For ciphers, the same degree of error correction has generally required use of computers.

Chapter 8

Cryptographic Hash Function



A cryptographic hash function (specifically, SHA-1) at work. Note that even small changes in the source input (here in the word "over") drastically change the resulting output, by the so-called avalanche effect.

A **cryptographic hash function** is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the (**cryptographic**) **hash value**, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is often called the "message," and the hash value is sometimes called the **message digest** or simply **digest.**

The ideal cryptographic hash function has four main or significant properties:

- it is easy to compute the hash value for any given message,
- it is infeasible to find a message that has a given hash,
- it is infeasible to modify a message without hash being changed,
- it is infeasible to find two different messages with the same hash.

Cryptographic hash functions have many information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information security contexts, cryptographic hash values are sometimes called (**digital**) **fingerprints, checksums,** or just **hash values,** even though all these terms stand for functions with rather different properties and purposes.

Properties

Most cryptographic hash functions are designed to take a string of any length as input and produce a fixed-length hash value.

A cryptographic hash function must be able to withstand all known types of cryptanalytic attack. As a minimum, it must have the following properties:

- *Preimage resistance*

  Given a hash $h$it should be difficult to find any message $m$such that $h = hash(m)$. This concept is related to that of one-way function. Functions that lack this property are vulnerable to preimage attacks.

- *Second preimage resistance*

  Given an input $m_1$it should be difficult to find another input m_2— where $m_1 \neq m_2$— such that $hash(m_1) = hash(m_2)$. This property is sometimes referred to as *weak collision resistance,* and functions that lack this property are vulnerable to second preimage attacks.

- *Collision resistance*

  It should be difficult to find two different messages $m_1$and $m_2$such that $hash(m_1) = hash(m_2)$. Such a pair is called a cryptographic hash collision. This property is sometimes referred to as *strong collision resistance.* It requires a hash value at least twice as long as that required for preimage-resistance, otherwise collisions may be found by a birthday attack.

These properties imply that a malicious adversary cannot replace or modify the input data without changing its digest. Thus, if two strings have the same digest, one can be very confident that they are identical.

A function meeting these criteria may still have undesirable properties. Currently popular cryptographic hash functions are vulnerable to *length-extension* attacks: given $h(m)$ and $len(m)$ but not m, by choosing a suitable m' an attacker can calculate $h(m||m')$ where | | denotes concatenation. This property can be used to break naive authentication schemes based on hash functions. The HMAC construction works around these problems.

Ideally, one may wish for even stronger conditions. It should be impossible for an adversary to find two messages with substantially similar digests; or to infer any useful information about the data, given only its digest. Therefore, a cryptographic hash function should behave as much as possible like a random function while still being deterministic and efficiently computable.

Checksum algorithms, such as CRC32 and other cyclic redundancy checks, are designed to meet much weaker requirements, and are generally unsuitable as cryptographic hash functions. For example, a CRC was used for message integrity in the WEP encryption standard, but an attack was readily discovered which exploited the linearity of the checksum.

Degree of difficulty

In cryptographic practice, "difficult" generally means "almost certainly beyond the reach of any adversary who must be prevented from breaking the system for as long as the security of the system is deemed important." The meaning of the term is therefore somewhat dependent on the application, since the effort that a malicious agent may put into the task is usually proportional to his expected gain. However, since the needed effort usually grows very quickly with the digest length, even a thousand-fold advantage in processing power can be neutralized by adding a few dozen bits to the latter.

In some theoretical analyses "difficult" has a specific mathematical meaning, such as *not solvable in asymptotic polynomial time.* Such interpretations of *difficulty* are important in the study of provably secure cryptographic hash functions but do not usually have a strong connection to practical security. For example, an exponential time algorithm can sometimes still be fast enough to make a feasible attack. Conversely, a polynomial time algorithm (e.g. one that requires n^{20} steps for *n*-digit keys) may be too slow for any practical use.

Illustration

An illustration of the potential use of a cryptographic hash is as follows: Alice poses a tough math problem to Bob, and claims she has solved it. Bob would like to try it himself, but would yet like to be sure that Alice is not bluffing. Therefore, Alice writes

down her solution, appends a random nonce, computes its hash and tells Bob the hash value (whilst keeping the solution and nonce secret). This way, when Bob comes up with the solution himself a few days later, Alice can prove that she had the solution earlier by revealing the nonce to Bob. (This is an example of a simple commitment scheme; in actual practice, Alice and Bob will often be computer programs, and the secret would be something less easily spoofed than a claimed puzzle solution).

Applications

Verifying the integrity of files or messages

An important application of secure hashes is verification of message integrity. Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event).

For this reason, most digital signature algorithms only confirm the authenticity of a hashed digest of the message to be "signed." Verifying the authenticity of a hashed digest of the message is considered proof that the message itself is authentic.

A related application is password verification. Passwords are usually not stored in cleartext, for obvious reasons, but instead in digest form. To authenticate a user, the password presented by the user is hashed and compared with the stored hash.

File or data identifier

A message digest can also serve as a means of reliably identifying a file; several source code management systems, including Git, Mercurial and Monotone, use the sha1sum of various types of content (file content, directory trees, ancestry information, etc) to uniquely identify them. Hashes are used to identify files on peer-to-peer filesharing networks. For example, in an ed2k link, an MD4-variant hash is combined with the file size, providing sufficient information for locating file sources, downloading the file and verifying its contents. Magnet links are another example. Such file hashes are often the top hash of a hash list or a hash tree which allows for additional benefits.

One of the main applications of a hash function is to allow the fast look-up of a data in a hash table. Being hash functions of a particular kind, cryptographic hash functions lend themselves well to this application too.

However, compared with standard hash functions, cryptographic hash functions tend to be much more expensive computationally. For this reason, they tend to be used in contexts where it is necessary for users to protect themselves against the possibility of forgery (the creation of data with the same digest as the expected data) by potentially malicious participants.

Pseudorandom generation and key derivation

Hash functions can also be used in the generation of pseudorandom bits, or to derive new keys or passwords from a single, secure key or password.

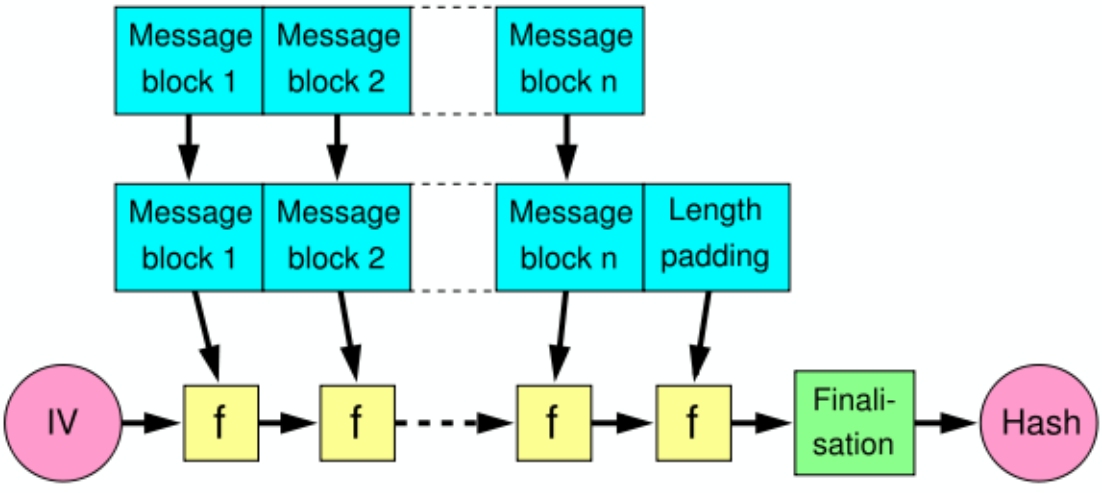
Hash functions based on block ciphers

There are several methods to use a block cipher to build a cryptographic hash function, specifically a one-way compression function.

The methods resemble the block cipher modes of operation usually used for encryption. All well-known hash functions, including MD4, MD5, SHA-1 and SHA-2 are built from block-cipher-like components designed for the purpose, with feedback to ensure that the resulting function is not bijective. SHA-3 finalists include functions with block-cipher-like components (e.g., Skein, BLAKE) and functions based on other designs (e.g., JH, Keccak).

A standard block cipher such as AES can be used in place of these custom block ciphers; that might be useful when an embedded system needs to implement both encryption and hashing with minimal code size or hardware area. However, that approach can have costs in efficiency and security. The ciphers in hash functions are built for hashing: they use large keys and blocks, can efficiently change keys every block, and have been designed and vetted for resistance to related-key attacks. General-purpose ciphers tend to have different design goals. In particular, AES has key and block sizes that make it nontrivial to use to generate long hash values; AES encryption becomes less efficient when the key changes each block; and related-key attacks make it potentially less secure for use in a hash function than for encryption.

Merkle–Damgård construction



The Merkle–Damgård hash construction.

A hash function must be able to process an arbitrary-length message into a fixed-length output. This can be achieved by breaking the input up into a series of equal-sized blocks, and operating on them in sequence using a one-way compression function. The compression function can either be specially designed for hashing or be built from a block cipher. A hash function built with the Merkle–Damgård construction is as resistant to collisions as is its compression function; any collision for the full hash function can be traced back to a collision in the compression function.

The last block processed should also be unambiguously length padded; this is crucial to the security of this construction. This construction is called the Merkle–Damgård construction. Most widely used hash functions, including SHA-1 and MD5, take this form.

The construction has certain inherent flaws, including length-extension and generate-and-paste attacks, and cannot be parallelized. As a result, many entrants in the current NIST hash function competition are built on different, sometimes novel, constructions.

Use in building other cryptographic primitives

Hash functions can be used to build other cryptographic primitives. For these other primitives to be cryptographically secure, care must be taken to build them correctly.

Message authentication codes (MACs) (also called keyed hash functions) are often built from hash functions. HMAC is such a MAC.

Just as block ciphers can be used to build hash functions, hash functions can be used to build block ciphers. Luby-Rackoff constructions using hash functions can be provably secure if the underlying hash function is secure. Also, many hash functions (including SHA-1 and SHA-2) are built by using a special-purpose block cipher in a Davies-Meyer or other construction. That cipher can also be used in a conventional mode of operation, without the same security guarantees.

Pseudorandom number generators (PRNGs) can be built using hash functions. This is done by combining a (secret) random seed with a counter and hashing it.

Some hash functions, such as Skein, Keccak, and RadioGatún output an arbitrarily long stream and can be used as a stream cipher, and stream ciphers can also be built from fixed-length digest hash functions. Often this is done by first building a cryptographically secure pseudorandom number generator and then using its stream of random bytes as keystream. SEAL is a stream cipher that uses SHA-1 to generate internal tables, which are then used in a keystream generator more or less unrelated to the hash algorithm. SEAL is not guaranteed to be as strong (or weak) as SHA-1.

Concatenation of cryptographic hash functions

Concatenating outputs from multiple hash functions provides collision resistance as good as the strongest of the algorithms included in the concatenated result. For example, older versions of TLS/SSL use concatenated MD5 and SHA-1 sums; that ensures that a method to find collisions in one of the functions doesn't allow forging traffic protected with both functions.

For Merkle-Damgård hash functions, the concatenated function is as collision-resistant as its strongest component, but not more collision-resistant. Joux noted that 2-collisions lead to n-collisions: if it is feasible to find two messages with the same MD5 hash, it is effectively no more difficult to find as many messages as the attacker desires with identical MD5 hashes. Among the n messages with the same MD5 hash, there is likely to be a collision in SHA-1. The additional work needed to find the SHA-1 collision (beyond the exponential birthday search) is polynomial. This argument is summarized by Finney. A more current paper and full proof of the security of such a combined construction gives a clearer and more complete explanation of the above.

Cryptographic hash algorithms

There is a long list of cryptographic hash functions, although many have been found to be vulnerable and should not be used. Even if a hash function has never been broken, a successful attack against a weakened variant thereof may undermine the experts' confidence and lead to its abandonment. For instance, in August 2004 weaknesses were found in a number of hash functions that were popular at the time, including SHA-0, RIPEMD, and MD5. This has called into question the long-term security of later algorithms which are derived from these hash functions — in particular, SHA-1 (a strengthened version of SHA-0), RIPEMD-128, and RIPEMD-160 (both strengthened versions of RIPEMD). Neither SHA-0 nor RIPEMD are widely used since they were replaced by their strengthened versions.

As of 2009, the two most commonly used cryptographic hash functions are MD5 and SHA-1. However, MD5 has been broken; an attack against it was used to break SSL in 2008.

The SHA-0 and SHA-1 hash functions were developed by the NSA. In February 2005, a successful attack on SHA-1 was reported, finding collisions in about 2^{69} hashing operations, rather than the 2^{80} expected for a 160-bit hash function. In August 2005, another successful attack on SHA-1 was reported, finding collisions in 2^{63} operations. Theoretical weaknesses of SHA-1 exist as well, suggesting that it may be practical to break within years. New applications can avoid these problems by using more advanced members of the SHA family, such as SHA-2, or using techniques such as randomized hashing that do not require collision resistance.

However, to ensure the long-term robustness of applications that use hash functions, there is a competition to design a replacement for SHA-2, which will be given the name SHA-3 and become a FIPS standard around 2012.

Algorithm	Output size (bits)	Internal state size	Block size	Length size	Word size	Collision attacks (complexity)	Preimage attacks (complexity)
GOST	256	256	256	256	32	Yes 2^{105}	Yes 2^{192}
HAVAL	256/224/192/160/128	256	1024	64	32	Yes	
MD2	128	384	128	No	32	Yes $2^{63.3}$	Yes 2^{73}
MD4	128	128	512	64	32	Yes 3	Yes $2^{70.4}$
MD5	128	128	512	64	32	$2^{20.96}$	Yes $2^{123.4}$
PANAMA	256	8736	256	No	32	Yes	
RadioGatún	up to 608/1216 (19 words)	58 words	3 words	No	1-64	With flaws (2^{352} or 2^{704})	
RIPEMD	128	128	512	64	32	Yes 2^{18}	
RIPEMD-128/256	128/256	128/256	512	64	32	No	
RIPEMD-160/320	160/320	160/320	512	64	32	No	
SHA-0	160	160	512	64	32	Yes $2^{33.6}$	
SHA-1	160	160	512	64	32	2^{51}	No
SHA-256/224	256/224	256	512	64	32	No	No
SHA-512/384	512/384	512	1024	128	64	No	No
Tiger(2)-192/160/128	192/160/128	192	512	64	64	2^{62}:19	Yes $2^{184.3}$
WHIRLPOOL	512	512	512	256	8	No	

Note: The *internal state* here means the "internal hash sum" after each compression of a data block. Most hash algorithms also internally use some additional variables such as length of the data compressed so far since that is needed for the length padding in the end.

Chapter 9

Deniable Encryption

In cryptography and steganography, **deniable encryption** is encryption that allows its users to convincingly deny that the data is encrypted, or that they are able to decrypt it. Such convincing denials may or may not be genuine. For example, although suspicions might exist that the data is encrypted, it may be impossible to prove it without the cooperation of the users. If the data is encrypted, the users genuinely may not be able to decrypt it. Deniable encryption serves to undermine an attacker's confidence either that data is encrypted, or that the person in possession of it can decrypt it and provide the associated plaintext.

Normally ciphertexts decrypt to a single plaintext and hence once decrypted, the encryption user cannot claim that he encrypted a different message. Deniable encryption allows its users to decrypt the ciphertext to produce a different (innocuous but plausible) plaintext and insist that it is what they encrypted. The holder of the ciphertext will not have the means to differentiate between the true plaintext, and the bogus-claim plaintext.

Function

Deniable encryption allows an encrypted message to be decrypted to different sensible plaintexts, depending on the key used, or otherwise makes it impossible to prove the existence of the real message without the proper encryption key. This allows the sender to have plausible deniability if compelled to give up his or her encryption key. The notion of "deniable encryption" was used by Julian Assange and Ralf Weinmann in the Rubberhose filesystem and explored in detail in a paper by Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky in 1996.

Scenario

Deniable encryption allows the sender of an encrypted message to deny sending that message. This requires a trusted third party. A possible scenario works like this:

1. Alice is the wife of Bob, who suspects his wife is engaged in adultery. She wants to communicate with her secret lover Carl. She creates two keys, one intended to

be kept secret, the other intended to be sacrificed. She passes the secret key (or both) to Carl.
2. Alice constructs an innocuous message M1 for Carl (intended to be revealed to Bob in case of discovery) and an incriminating love letter M2 to Carl. She constructs a cipher-text C out of both messages M1, M2 and emails it to Carl.
3. Carl uses his key to decrypt M2 (and possibly M1, in order to read the fake message too).
4. Bob finds out about the email to Carl, becomes suspicious and forces Alice to decrypt the message.
5. Alice uses the sacrificial key and reveals the innocuous message M1 to Bob. Since Bob does not know the other key, he has to assume that there is no other message M2.

Another possible scenario involves Alice sending the same ciphertext to Bob and Carl, to whom she has sent different keys. Bob's key could decrypt the ciphertext to a claim that Carl has maligned Bob, and Carl's key could decrypt the ciphertext to a claim that Bob has maligned Carl. Under the assumption that neither Bob nor Carl can find out each other's keys, Alice's ruse would never be discovered unless Bob and Carl find out that they have different keys.

Modern forms of deniable encryption

Modern deniable encryption techniques exploit the pseudorandom permutation properties of existing block ciphers, making it cryptographically infeasible to prove that the ciphertext is not random data generated by a cryptographically secure pseudorandom number generator. This is used in combination with some decoy data that the user would plausibly want to keep confidential that will be revealed to the attacker, claiming that this is all there is. This form of deniable encryption is sometimes referred to as steganography.

One example of deniable encryption is a cryptographic filesystem that employs a concept of abstract "layers", where each layer would be decrypted with a different encryption key. Additionally, special "chaff layers" are filled with random data in order to have plausible deniability of the existence of real layers and their encryption keys. The user will store decoy files on one or more layers while denying the existence of others, claiming that the rest of space is taken up by chaff layers. Physically, these types of filesystems are typically stored in a single directory consisting of equal-length files with filenames that are either randomized (in case they belong to chaff layers), or cryptographic hashes of strings identifying the blocks. The timestamps of these files are always randomized. Examples of this approach include Rubberhose filesystem and PhoneBookFS.

Another approach utilized by some conventional disk encryption software suites is creating a second encrypted volume within a container volume. The container volume is first formatted by filling it with encrypted random data, and then initializing a filesystem on it. The user then fills some of the filesystem with legitimate, but plausible-looking decoy files that the user would seem to have an incentive to hide. Next, a new encrypted

volume (the hidden volume) is allocated within the free space of the container filesystem which will be used for data the user actually wants to hide. Since an adversary cannot differentiate between encrypted data and the random data used to initialize the outer volume, this inner volume is now undetectable. Concerns have, however, been raised for the level of plausible deniability in hiding information this way – the contents of the "outer" container filesystem (in particular the access or modification timestamps on the data stored) could raise suspicions as a result of being frozen in its initial state to prevent the user from corrupting the hidden volume. This problem can be eliminated by instructing the system not to protect the hidden volume, although this could result in lost data. FreeOTFE and BestCrypt can have many hidden volumes in a container; TrueCrypt is limited to one hidden volume.

The use of insecure block ciphers or weak pseudorandom number generators can make it possible to compromise the deniability of such filesystems. To eliminate the need to assume that the used pseudorandom number generation is cryptographically secure, it is advisable to instead fill the encrypted space with pseudo-random data which has itself been encrypted using a separate encryption key. Since encrypted pseudo-random data is impossible to differentiate from the sensitive data that the user wishes to protect, the presence of the hidden volume cannot be determined by an attacker

The flawed use of block cipher modes of operation (such as Cipher block chaining with predictable IVs) can also compromise a hidden volume due to watermarking attacks.

Malleable encryption

Some in-transit encrypted messaging suites, such as Off-the-Record Messaging, offer malleable encryption which gives the participants plausible deniability of their conversations. While malleable encryption is not technically "deniable encryption" in that its ciphertexts do not decrypt into multiple plaintexts, its deniability refers to the inability of an adversary to prove that the participants had a conversation or said anything in particular.

This is achieved by the fact that all information necessary to forge messages is appended to the encrypted messages – if an adversary is able to create digitally authentic messages in a conversation, he is also able to forge messages in the conversation. This is used in conjunction with perfect forward secrecy to assure that the compromise of encryption keys of individual messages does not compromise additional conversations or messages.

Software

- OpenPuff, freeware semi-open-source steganography for MS Windows.
- BestCrypt, commercial on-the-fly disk encryption for MS Windows.
- FreeOTFE, opensource on-the-fly disk encryption for MS Windows and PocketPC PDAs that provides both deniable encryption and plausible deniability. Offers an extensive range of encryption options, and doesn't need to be installed before use.

- Off-the-Record Messaging, a cryptographic technique providing true deniability for instant messaging.
- PhoneBookFS, another cryptographic filesystem for Linux, providing plausible deniability through chaff and layers. A FUSE implementation. No longer maintained.
- rubberhose. Defunct project (Last release in 2000, not compatible with modern Linux distributions)
- StegFS, the current successor to the ideas embodied by the Rubberhose and PhoneBookFS filesystems
- TrueCrypt, which is on-the-fly disk encryption software for Windows, Mac and Linux that provides limited deniable encryption and to some extent (due to limitations on the number of hidden volumes which can be created) plausible deniability, and doesn't need to be installed before use as long as the user has full administrator rights
- Vanish - a research prototype implementation of self-destructing data storage
- ScramDisk 4 Linux - A free software suite of tools, for GNU/Linux systems, which can open and create scramdisk and truecrypt container.

Chapter 10

Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985.

Elliptic curves are also used in several integer factorization algorithms that have applications in cryptography, such as Lenstra elliptic curve factorization.

Introduction

Public-key cryptography is based on the intractability of certain mathematical problems. Early public-key systems, such as the RSA algorithm, are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly-known base point is unfeasible. The size of the elliptic curve determines the difficulty of the problem. It is believed that the same level of security afforded by an RSA-based system with a large modulus can be achieved with a much smaller elliptic curve group. Using a small group reduces storage and transmission requirements.

For current cryptographic purposes, an *elliptic curve* is a plane curve which consists of the points satisfying the equation

$$y^2 = x^3 + ax + b,$$

along with a distinguished point at infinity, denoted ∞. (The coordinates here are to be chosen from a fixed finite field of characteristic not equal to 2 or 3, or the curve equation will be somewhat more complicated.) This set together with the group operation of the elliptic group theory form an Abelian group, with the point at infinity as identity element.

The structure of the group is inherited from the divisor group of the underlying algebraic variety.

As for other popular public key cryptosystems, no mathematical proof of security has been published for ECC as of 2009. However, the U.S. National Security Agency has endorsed ECC by including schemes based on it in its Suite B set of recommended algorithms and allows their use for protecting information classified up to top secret with 384-bit keys. While the RSA patent expired in 2000, there are patents in force covering certain aspects of ECC technology, though some argue that the Federal elliptic curve digital signature standard (ECDSA; NIST FIPS 186-3) and certain practical ECC-based key exchange schemes (including ECDH) can be implemented without infringing them.

Cryptographic premise

The entire security of ECC depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points.

Cryptographic schemes

Several discrete logarithm-based protocols have been adapted to elliptic curves, replacing the group $(\mathbb{Z}_p)^\times$ with an elliptic curve:

- the elliptic curve Diffie–Hellman key agreement scheme is based on the Diffie–Hellman scheme,
- the Elliptic Curve Integrated Encryption Scheme (ECIES), also known as Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme,
- the Elliptic Curve Digital Signature Algorithm is based on the Digital Signature Algorithm,
- the ECMQV key agreement scheme is based on the MQV key agreement scheme.

At the RSA Conference 2005, the National Security Agency (NSA) announced Suite B which exclusively uses ECC for digital signature generation and key exchange. The suite is intended to protect both classified and unclassified national security systems and information.

Recently, a large number of cryptographic primitives based on bilinear mappings on various elliptic curve groups, such as the Weil and Tate pairings, have been introduced. Schemes based on these primitives provide efficient identity-based encryption as well as pairing-based signatures, signcryption, key agreement, and proxy re-encryption.

Domain parameters

To use ECC all parties must agree on all the elements defining the elliptic curve, that is, the *domain parameters* of the scheme. The field is defined by p in the prime case and the pair of m and f in the binary case. The elliptic curve is defined by the constants a and b

used in its defining equation. Finally, the cyclic subgroup is defined by its *generator* (aka. *base point*) *G*. For cryptographic application the order of *G*, that is the smallest non-negative number *n* such that $nG = \infty$, is normally prime. Since *n* is the size of a subgroup of $E(\mathbb{F}_p)$ it follows from Lagrange's theorem that the number $h = \frac{|E|}{n}$ is an integer. In cryptographic applications this number *h*, called the *cofactor*, must be small ($h \leq 4$) and, preferably, *h* = 1. Let us summarize: in the prime case the domain parameters are (*p*,*a*,*b*,*G*,*n*,*h*) and in the binary case they are (*m*,*f*,*a*,*b*,*G*,*n*,*h*).

Unless there is an assurance that domain parameters were generated by a party trusted with respect to their use, the domain parameters *must* be validated before use.

The generation of domain parameters is not usually done by each participant since this involves counting the number of points on a curve which is time-consuming and troublesome to implement. As a result several standard bodies published domain parameters of elliptic curves for several common field sizes:

- NIST, Recommended Elliptic Curves for Government Use
- SECG, SEC 2: Recommended Elliptic Curve Domain Parameters

Test vectors are also available.

If one (despite the said above) wants to build one's own domain parameters one should select the underlying field and then use one of the following strategies to find a curve with appropriate (i.e., near prime) number of points using one of the following methods:

- select a random curve and use a general point-counting algorithm, for example, Schoof's algorithm or Schoof–Elkies–Atkin algorithm,
- select a random curve from a family which allows easy calculation of the number of points (e.g., Koblitz curves), or
- select the number of points and generate a curve with this number of points using *complex multiplication* technique.

Several classes of curves are weak and should be avoided:

- curves over $\mathbb{F}_{2^m}$ with non-prime *m* are vulnerable to Weil descent attacks.
- curves such that *n* divides $p^B - 1$ (where *p* is the characteristic of the field – *q* for a prime field, or 2 for a binary field) for sufficiently small *B* are vulnerable to MOV attack which applies usual DLP in a small degree extension field of $\mathbb{F}_p$ to solve ECDLP. The bound *B* should be chosen so that discrete logarithms in the field $\mathbb{F}_{p^B}$ are at least as difficult to compute as discrete logs on the elliptic curve $E(\mathbb{F}_q)$.
- curves such that $|E(\mathbb{F}_q)| = q$ are vulnerable to the attack that maps the points on the curve to the additive group of $\mathbb{F}_q$

Key sizes

Since all the fastest known algorithms that allow to solve the ECDLP (baby-step giant-step, Pollard's rho, etc.), need $O(\sqrt{n})$steps, it follows that the size of the underlying field shall be roughly twice the security parameter. For example, for 128-bit security one needs a curve over $\mathbb{F}_q$, where $q \approx 2^{256}$. This can be contrasted with finite-field cryptography (e.g., DSA) which requires 3072-bit public keys and 256-bit private keys, and integer factorization cryptography (e.g., RSA) which requires 3072-bit public and private keys.

The hardest ECC scheme (publicly) broken to date had a 112-bit key for the prime field case and a 109-bit key for the binary field case. For the prime field case this was broken in July 2009 using a cluster of over 200 PlayStation 3 game consoles and could have been finished in 3.5 months using this cluster when running continuously. For the binary field case, it was broken in April 2004 using 2600 computers for 17 months.

A current project is aiming at breaking the ECC2K-130 challenge by Certicom, by using a wide range of different hardware : CPUs, GPUs, FPGA.

Projective coordinates

A close examination of the addition rules shows that in order to add two points one needs not only several additions and multiplications in $\mathbb{F}_q$but also an inversion operation. The inversion (for given $x \in \mathbb{F}_q$find $y \in \mathbb{F}_q$such that $xy = 1$) is one to two orders of magnitude slower than multiplication. Fortunately, points on a curve can be represented in different coordinate systems which do not require an inversion operation to add two points. Several such systems were proposed: in the *projective* system each point is represented by three coordinates (X,Y,Z) using the following relation: $x = \frac{X}{Z}, y = \frac{Y}{Z}$; in the *Jacobian system* a point is also represented with three coordinates (X,Y,Z), but a different relation is used: $x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}$; in the *López–Dahab system* the relation is $x = \frac{X}{Z}, y = \frac{Y}{Z^2}$; in the *modified Jacobian* system the same relations are used but four coordinates are stored and used for calculations (X,Y,Z,aZ^4); and in the *Chudnovsky Jacobian* system five coordinates are used (X,Y,Z,Z^2,Z^3). Note that there may be different naming conventions, for example, IEEE P1363-2000 standard uses "projective coordinates" to refer to what is commonly called Jacobian coordinates. An additional speed-up is possible if mixed coordinates are used.

Fast reduction (NIST curves)

Reduction modulo p (which is needed for addition and multiplication) can be executed much faster if the prime p is a pseudo-Mersenne prime that is $p \approx 2^d$, for example, $p = 2^{521} - 1$ or $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Compared to Barrett reduction there can be an order of magnitude speedup. The speedup here is a practical rather than theoretical one, and derives from the fact that the moduli of numbers against numbers near powers of two can be performed efficiently by computers operating on binary numbers with bitwise operations.

The curves over $\mathbb{F}_p$ with pseudo-Mersenne p are recommended by NIST. Yet another advantage of the NIST curves is the fact that they use $a = -3$ which improves addition in Jacobian coordinates.

NIST-recommended elliptic curves

NIST recommends fifteen elliptic curves. Specifically, FIPS 186-3 has ten recommended finite fields:

- Five prime fields $\mathbb{F}_p$ for certain primes p of sizes 192, 224, 256, 384, and 521 bits. For each of the prime fields, one elliptic curve is recommended.
- Five binary fields $\mathbb{F}_{2^m}$ for m equal 163, 233, 283, 409, and 571. For each of the binary fields, one elliptic curve and one Koblitz curve was selected.

The NIST recommendation thus contains a total of five prime curves and ten binary curves. The curves were chosen for optimal security and implementation efficiency.

Side-channel attacks

Unlike DLP systems (where it is possible to use the same procedure for squaring and multiplication) the EC addition is significantly different for doubling ($P = Q$) and general addition ($P \neq Q$) depending on the coordinate system used. Consequently, it is important to counteract side channel attacks (e.g., timing or simple/differential power analysis attacks) using, for example, fixed pattern window (aka. comb) methods (note that this does not increase the computation time). Another concern for ECC-systems is the danger of fault attacks, especially when running on smart cards (see, for example, Biehl et al.).

Quantum computing attacks

Elliptic curve cryptography is vulnerable to a modified Shor's algorithm for solving the discrete logarithm problem on elliptic curves.

Chapter 11

Lamport Signature

In cryptography, a **Lamport signature** or **Lamport one-time signature scheme** is a method for constructing a digital signature. Lamport signatures can be built from any cryptographically secure one-way function; usually a cryptographic hash function is used.

Although the potential development of quantum computers threatens the security of many common forms of cryptography such as RSA, it is believed that Lamport signatures with large hash functions would still be secure in that event. Unfortunately, each Lamport key can only be used to sign a single message. However, combined with hash trees, a single key could be used for many messages, making this a fairly efficient digital signature scheme.

The Lamport signature cryptosystem was invented in 1979 by Leslie Lamport, named after the inventor.

Example

Alice has a 256-bit cryptographic hash function and some kind of secure random number generator. She wants to create and use a Lamport key pair, that is, a private key and a corresponding public key.

Making the key pair

To create the private key Alice uses the random number generator to produce 256 pairs of random numbers (2×256 numbers in total), each number being 256 bits in size, that is, a total of 2×256×256 bits = 16 KiB in total. This is her private key and she will store it away in a secure place for later use.

To create the public key she hashes each of the 512 random numbers in the private key, thus creating 512 hashes, each 256 bits in size. (Also 16 KiB in total.) These 512 numbers form her public key, which she will share with the world.

Signing the message

Later Alice wants to sign a message. First she hashes the message to a 256-bit hash sum. Then for each bit in the hash sum she picks the corresponding number from her private key. If for instance the first bit in the hash sum is a 0, she picks the *first* number in the first pair. If the first bit instead is a 1, then she uses the *second* number in the first pair. And so on. This gives her 256 random numbers. That is, 256×256 bits = 8 KiB in total. These random numbers are her signature and she publishes them along with the message.

Note that now Alice's private key is used and should never be used again. The other 256 random numbers that she did not use for the signature she must never publish or use. Preferably she should delete them. Otherwise others will later be able to create false signatures.

Verifying the signature

Then Bob wants to verify Alice's signature of the message. He also hashes the message to get a 256-bit hash sum. Then he uses the bits in the hash sum to pick out 256 of the hashes in Alice's public key. He picks the hashes in the same manner that Alice picked the random numbers for the signature. That is, if the first bit of the message hash is a 0, he picks the *first* hash in the first pair, and so on.

Then Bob hashes each of the 256 random numbers in Alice's signature. This gives him 256 hashes. If these 256 hashes exactly match the 256 hashes he just picked from Alice's public key then the signature is ok. If not, then something is wrong.

Note that prior to that Alice has published the signature of a message no one else knows the 2×256 random numbers in the private key. Thus no one else can create the proper list of 256 random numbers for the signature. And after Alice has published the signature others still do not know the other 256 random numbers and thus can not create signatures that fits other message hashes.

Mathematical description

Below is a short description of how Lamport signatures work, written in mathematical notation. Note that the "message" in this description is a fixed sized block of reasonable size, possibly (but not necessarily) the hash result of an arbitrary long message being signed.

Keys

Let k be a positive integer and let $P = \{0,1\}^k$ be the set of messages. Let $f : Y \to Z$ be a one-way function.

For $1 \leq i \leq k$ and $j \in \{0, 1\}$ the signer chooses $y_{i,j} \in Y$ randomly and computes $z_{i,j} = f(y_{i,j})$.

The private key K consists of $2k$ values $y_{i,j}$. The public key consists of the $2k$ values $z_{i,j}$.

Signing a message

Let $m = m_1 \dots m_k \in \{0, 1\}^k$ be a message.

The signature of the message is
$sig(m_1 \dots m_k) = (y_{1,m_1}, \dots, y_{k,m_k}) = (s_1, \dots, s_k)$.

Verifying a signature

The verifier validates a signature by checking that $f(s_i) = z_{i,m_i}$ for all $1 \leq i \leq k$.

In order to forge a message Eve would have to invert the one-way function f. This is assumed to be intractable for suitably sized inputs and outputs.

Security parameters

The security of Lamport signatures is based on security of the one way hash function, the length of its output and the quality of the input.

For a hash function that generates an n-bit message digest, the ideal preimage and 2nd preimage resistance on a single hash function invocation implies on the order of 2^n operations and 2^n bits of memory effort to find a collision under a classical computing model. According to Grover's algorithm, finding a preimage collision on a single invocation of an ideal hash function is upper bound on $O(2^{n/2})$ operations under a quantum computing model. In Lamport signatures, each bit of the public key and signature is based on short messages requiring only a single invocation to a hash function.

For each private key $y_{i,j}$ and its corresponding $z_{i,j}$ public key pair, the private key length must be selected so performing a preimage attack on the length of the input is not faster than performing a preimage attack on the length of the output. For example, in a degenerate case, if each private key $y_{i,j}$ element was only 16 bits in length, it is trivial to exhaustively search all 2^{16} possible private key combinations in 2^{15} operations to find a match with the output, irrespective of the message digest length. Therefore a balanced system design ensures both lengths are approximately equal.

Based on Grover's algorithm, a quantum secure system, the length of the public key elements $z_{i,j}$, the private key elements $y_{i,j}$ and the signature elements $s_{i,j}$ must be no less than 2 times larger than the security rating of the system. That is:

- A 80-bit secure system uses element lengths of no less than 160 bit;
- A 128-bit secure systems uses element lengths of no less than 256 bit;

However caution should be taken as the idealistic work estimates above assume an ideal (perfect) hash function and are limited to attacks that target only a single preimage at a time. It is known under a conventional computing model that if $2^{3n/5}$ preimages are searched, the full cost per preimage decreases from $2^{n/2}$ *to* $2^{2n/5}$. *Selecting the optimum element size taking into account the collection of multiple message digests is an open problem. Selection of larger element sizes and stronger hash functions, such as 512-bit elements and SHA-512, ensures greater security margins to manage these unknowns.*

Optimisations and variants

Short private key

Instead of creating and storing all the random numbers of the private key a single key of sufficient size can be stored. (Usually the same size as one of the random numbers in the private key.) The single key can then be used as the seed for a cryptographically secure pseudorandom number generator (CSPRNG) to create all the random numbers in the private key when needed.

In the same manner a single key can be used together with a CSPRNG to create many Lamport keys. Preferably then some kind of random access CSPRNG should be used, such as BBS.

Short public key

A Lamport signature can be combined with a hash list, making it possible to only publish a single hash instead of all the hashes in the public key. That is, instead of the $2k$ values z_{ij}. To be able to verify the random numbers in a signature against the single hash all the unused hashes in the public key then needs to be included in the signatures, resulting in signatures of about twice the size, but results in significantly shorter public keys. That is, the values (z_{ij}) for all $j \neq m_i$needs to be included.

Public key for multiple messages

Each Lamport public key can only be used to sign one single message, which means many keys have to be published if many messages are to be signed. But a hash tree can be used on those public keys, publishing the top hash of the hash tree instead. This increases the size of the resulting signature, since parts of the hash tree have to be included in the signature, but it makes it possible to publish a single hash that then can be used to verify any given number of future signatures.

Hashing the message

Unlike some other signature schemes (e.g. RSA) the Lamport signature scheme does not require that the message *m* is hashed before it is signed. A system for signing long messages can use a collision resistant hash function h and sign h(*m*) instead of *m*.

Chapter 12

ID-Based Encryption

ID-based encryption (or **Identity-Based Encryption (IBE)**) is an important primitive of ID-based cryptography. As such it is a type of public-key encryption in which the public key of a user is some unique information about the identity of the user (e.g. a user's email address). This can use the text-value of the name or domain name as a key or the physical IP address it translates to.

The first implementation of an email-address based PKI was developed by Adi Shamir in 1984, which allowed users to verify digital signatures using only public information such as the user's identifier.

ID-based encryption was proposed by Adi Shamir in 1984. He was however only able to give an instantiation of identity-based signatures. Identity-based encryption remained an open problem for many years. One example of the research leading up to identity-based encryption is provided in Maurer.

The Boneh/Franklin's pairing-based encryption scheme and Cocks's encryption scheme based on quadratic residues both solved the IBE problem in 2001.

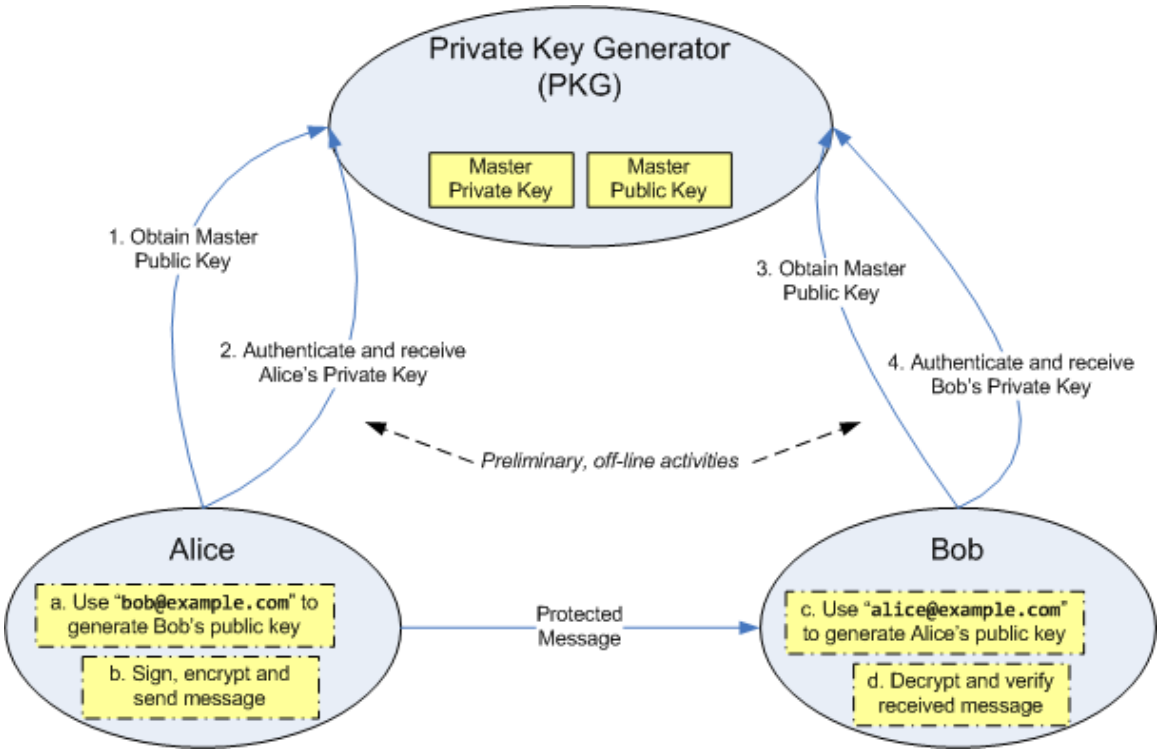
Usage

Identity-based systems allow any party to generate a public key from a known identity value such as an ASCII string. A trusted third party, called the Private Key Generator (PKG), generates the corresponding private keys. To operate, the PKG first publishes a master public key, and retains the corresponding **master private key** (referred to as *master key*). Given the master public key, any party can compute a public key corresponding to the identity *ID* by combining the master public key with the identity value. To obtain a corresponding private key, the party authorized to use the identity *ID* contacts the PKG, which uses the master private key to generate the private key for identity *ID*.

As a result, parties may encrypt messages (or verify signatures) with no prior distribution of keys between individual participants. This is extremely useful in cases where pre-distribution of authenticated keys is inconvenient or infeasible due to technical restraints.

However, to decrypt or sign messages, the authorized user must obtain the appropriate private key from the PKG. A caveat of this approach is that the PKG must be highly trusted, as it is capable of generating any user's private key and may therefore decrypt (or sign) messages without authorization. Because any user's private key can be generated through the use of the third party's secret, this system has inherent key escrow. A number of variant systems have been proposed which remove the escrow including certificate-based encryption, secure key issuing cryptography and certificateless cryptography.

The steps involved are depicted in this diagram:



ID Based Encryption: Offline and Online Steps

Protocol framework

Dan Boneh and Matthew K. Franklin defined a set of four algorithms that form a complete IBE system:

- **Setup**: This algorithm is run by the PKG one time for creating the whole IBE environment. The master key is kept secret and used to derive users' private keys, while the system parameters are made public. It accepts a security parameter k(i.e. binary length of key material) and outputs:

1. A set $\mathcal{P}$of system parameters, including the message space and ciphertext space $\mathcal{M}$and $\mathcal{C}$,
2. a master key K_m.

- **Extract**: This algorithm is run by the PKG when a user requests his private key. Note that the verification of the authenticity of the requestor and the secure transport of $d$are problems with which IBE protocols do not try to deal. It takes as input $\mathcal{P}$, $K_m$and an identifier $ID \in \{0,1\}^*$and returns the private key $d$for user ID.

- **Encrypt**: Takes $\mathcal{P}$, a message $m \in \mathcal{M}$and $ID \in \{0,1\}^*$and outputs the encryption $c \in \mathcal{C}$.

- **Decrypt**: Accepts $d, \mathcal{P}$and $c \in \mathcal{C}$and returns $m \in \mathcal{M}$.

Correctness constraint

In order for the whole system to work, one has to postulate that:

$$\forall m \in \mathcal{M}, ID \in \{0,1\}^* : Decrypt\left(Extract\left(\mathcal{P}, K_m, ID\right), \mathcal{P}, Encrypt\left(\mathcal{P}, m, ID\right)\right) = m$$

Encryption schemes

The most efficient identity-based encryption schemes are currently based on bilinear pairings on elliptic curves, such as the Weil or Tate pairings. The first of these schemes was developed by Dan Boneh and Matthew K. Franklin (2001), and performs probabilistic encryption of arbitrary ciphertexts using an Elgamal-like approach. Though the Boneh-Franklin scheme is provably secure, the security proof rests on relatively new assumptions about the hardness of problems in certain elliptic curve groups.

Another approach to identity-based encryption was proposed by Clifford Cocks in 2001. The Cocks IBE scheme is based on well-studied assumptions (the quadratic residuosity assumption) but encrypts messages one bit at a time with a high degree of ciphertext expansion. Thus it is highly inefficient and impractical for sending all but the shortest messages, such as a session key for use with a symmetric cipher.

Advantages

One of the major advantages of any identity-based encryption scheme is that if there are only a finite number of users, after all users have been issued with keys the third party's secret can be destroyed. This can take place because this system assumes that, once issued, keys are always valid (as this basic system lacks a method of key revocation). The majority of derivatives of this system which have key revocation lose this advantage.

Moreover, as public keys are derived from identifiers, IBE eliminates the need for a public key distribution infrastructure. The authenticity of the public keys is guaranteed implicitly as long as the transport of the private keys to the corresponding user is kept secure (Authenticity, Integrity, Confidentiality).

Apart from these aspects, IBE offers interesting features emanating from the possibility to encode additional information into the identifier. For instance, a sender might specify an expiration date for a message. He appends this timestamp to the actual recipient's identity (possibly using some binary format like X.509). When the receiver contacts the PKG to retrieve the private key for this public key, the PKG can evaluate the identifier and decline the extraction if the expiration date has passed. Generally, embedding data in the ID corresponds to opening an additional channel between sender and PKG with authenticity guaranteed through the dependency of the private key on the identifier.

Drawbacks

- If a Private Key Generator (PKG) is compromised, all messages protected over the entire lifetime of the public-private key pair used by that server are also compromised. This makes the PKG a high value target to adversaries. To limit the exposure due to a compromised server, the master private-public key pair could be updated with a new independent key pair. However, this introduces a key-management problem where all users must have the most recent public key for the server.
- Because the Private Key Generator (PKG) generates private keys for users, it may decrypt and/or sign any message without authorisation. This implies that IBE systems cannot be used for non-repudiation. This may not be an issue for organizations that host their own PKG and are willing to trust their system administrators and do not require non-repudiation.
- The issue of implicit key escrow does not exist with the current PKI system wherein private keys are usually generated on the user's computer. Depending on the context key escrow can be seen as a positive feature (e.g., within Enterprises). A number of variant systems have been proposed which remove the escrow including certificate-based encryption, secret sharing, secure key issuing cryptography and certificateless cryptography.
- A secure channel between a user and the Private Key Generator (PKG) is required for transmitting the private key on joining the system. Here, a SSL-like connection is a common solution for a large-scale system. It is important to observe that users that hold accounts with the PKG must be able to authenticate themselves. In principle, this may be achieved through username,password or through public key pairs managed on smart cards.
- IBE solutions may rely on cryptographic techniques that are insecure against code breaking quantum computer attacks

Chapter 13

Commitment Scheme

In cryptography, a **commitment scheme** allows one to commit to a value while keeping it hidden, with the ability to reveal the committed value later. Commitments are used to bind a party to a value so that they cannot adapt to other messages in order to gain some kind of inappropriate advantage. They are important to a variety of cryptographic protocols including secure coin flipping, zero-knowledge proofs, and secure computation.

Interactions in a commitment scheme take place in two phases:

1. the *commit phase* during which a value is chosen and specified
2. the *reveal phase* during which the value is revealed and checked

In simple protocols, the commit phase consists of a single message from the sender to the receiver. This message is called *the commitment*. It is essential that the specific value chosen cannot be known by the receiver at that time (this is called the *hiding* property). A simple reveal phase would consist of a single message, *the opening*, from the sender to the receiver, followed by a check performed by the receiver. The value chosen during the commit phase must be the only one that the sender can compute and that validates during the reveal phase (this is called the *binding* property).

The concept of commitment schemes was first formalized by Gilles Brassard, David Chaum, and Claude Crepeau in 1988, but the concept was used without being treated formally prior to that. The notion of commitments appeared earliest in works by Manuel Blum, Shimon Even, and Shamir et al. The terminology seems to have been originated by Blum, although commitment schemes can be interchangeably called **bit commitment schemes**—sometimes reserved for the special case where the committed value is a binary bit.

Applications

Coin flipping

Suppose Alice and Bob want to resolve some dilemma via coin flipping. If they are physically in the same place, a typical procedure might be:

1. Alice "calls" the coin flip
2. Bob flips the coin
3. If Alice's call is correct, she wins, otherwise Bob wins

If they are not in the same place, this procedure is faulty. Alice would have to trust Bob's report of how the coin flip turned out, whereas Bob knows what result is more desirable for him. Using commitments, a similar procedure is:

1. Alice "calls" the coin flip and tells Bob only a *commitment* to her call,
2. Bob flips the coin and reports the result,
3. Alice reveals what she committed to
4. If Alice's revelation matches the coin result Bob reported, Alice wins

For Bob to be able to skew the results to his favor, he must be able to understand the call hidden in Alice's commitment. If the commitment scheme is a good one, Bob cannot skew the results. Similarly, Alice cannot affect the result if she cannot change the value she commits to.

A way to visualize a commitment scheme is to think of the sender as putting the value in a locked box, and giving the box to the receiver. The value in the box is hidden from the receiver, who cannot open the lock themselves. Since the receiver has the box, the value inside cannot be changed—merely revealed if the sender chooses to give them the key at some later time.

Zero-knowledge proofs

One particular motivating example is the use of commitment schemes in zero-knowledge proofs. Commitments are used in zero-knowledge proofs for two main purposes: first, to allow the prover to participate in "cut and choose" proofs where the verifier will be presented with a choice of what to learn, and the prover will reveal only what corresponds to the verifier's choice. Commitment schemes allow the prover to specify all the information in advance in a commitment, and only reveal what should be revealed later in the proof. Commitments are also used in zero-knowledge proofs by the verifier, who will often specify their choices ahead of time in a commitment. This allows zero-knowledge proofs to be composed in parallel without revealing additional information.

Verifiable secret sharing

Another important application of commitments is in verifiable secret sharing, a critical building block of secure multiparty computation. In a secret sharing scheme, each of several parties receive "shares" of a value that is meant to be hidden from everyone. If enough parties get together, their shares can be used to reconstruct the secret, but even a malicious cabal of insufficient size should learn nothing. Secret sharing is at the root of many protocols for secure computation: in order to securely compute a function of some shared input, the secret shares are manipulated instead. However, if shares are to be generated by malicious parties, it may be important that those shares can be checked for correctness. In a verifiable secret sharing scheme, the distribution of a secret is accompanied by commitments to the individual shares. The commitments reveal nothing that can help a dishonest cabal, but the shares allow each individual party to check to see if their shares are correct.

Defining the security of commitment schemes

Formal definitions of commitment schemes vary strongly in notation and in flavour. The first such flavour is whether the commitment scheme provides perfect or computational security with respect to the hiding or binding properties. Another such flavour is whether the commitment is interactive, i.e. whether both the commit phase and the reveal phase can be seen as being executed by a Cryptographic protocol or whether they are non-interactive, consisting of two algorithms *Commit* and *CheckReveal*. In the latter case *CheckReveal* can often be seen as a derandomised version of *Commit*, with the randomness used by *Commit* constituting the opening information.

If the commitment C to a value x is computed as $C:=Commit(x,open)$ with *open* the randomness used for computing the commitment, then $CheckReveal(C,x,open)$ simply consists in verifying the equation $C=Commit(x,open)$.

Using this notation and some knowledge about mathematical functions and probability theory we formalise different versions of the binding and hiding properties of commitments. The two most important combinations of these properties are perfectly binding and computationally hiding commitment schemes and computationally binding and perfectly hiding commitment schemes. Note that no commitment scheme can be at the same time perfectly binding and perfectly hiding.

Computational binding

Let *open* be chosen from a set of size 2^k, i.e., it can represented as a k bit string, and let $Commit_k$ be the corresponding commitment scheme. As the size of k determines the security of the commitment scheme it is called the security parameter.

Then for all non-uniform probabilistic polynomial time algorithms that output x,x' and *open*,*open*' of increasing length k, the probability that $x \neq x'$ and $Commit_k(x,open) = Commit_k(x',open')$ is a negligible function in k.

This is a form of Asymptotic analysis. It is also possible to state the same requirement using Concrete security: A commitment scheme *Commit* is (*t*,ε) secure, if for all algorithms that run in time *t* and output *x*,*x*',*open*,*open*' the probability that $x \neq x'$ and *Commit*(*x*,*open*) = *Commit*(*x*',*open*') is at most ε.

Perfect, statistical and computational hiding

Let U_k be the uniform distribution over the 2^k opening values for security parameter *k*. A commitment scheme is perfect, statistical, computational hiding, if for all $x \neq x'$ the probability ensembles $\{Commit_k(x, U_k)\}_{k\in\mathbb{N}}$ and $\{Commit_k(x', U_k)\}_{k\in\mathbb{N}}$ are equal, statistically close, or computationally indistinguishable.

Constructing commitment schemes

A commitment scheme can either be perfectly binding (it is impossible for Alice to alter her commitment after she has made it, even if she has unbounded computational resources) or perfectly concealing (it is impossible for Bob to find out the commitment without Alice revealing it, even if he has unbounded computational resources) but not both.

Bit-commitment from any one-way permutation (or injective one way function)

One can create a bit-commitment scheme from any one-way function that is injective. The scheme relies on the fact that every one-way function can be modified (via the Goldreich-Levin theorem) to possess a computationally hard-core predicate (while retaining the injective property). Let *f* be an injective one-way function, with *h* a hard-core predicate. Then to commit to a bit *b* Alice picks a random input *x* and sends the triple

$$(h, f(x), b \oplus h(x))$$

to Bob, where $\oplus$ denotes XOR, i.e. addition modulo 2. To decommit Alice simply sends *x* to Bob. Bob verifies by computing *f(x)* and comparing to the committed value. This scheme is concealing because for Bob to recover *b* he must recover *h(x)*. Since *h* is a computationally hard-core predicate, recovering *h(x)* from *f(x)* with probability greater than one-half is as hard as inverting *f*. Perfect binding follows from the fact that *f* is injective and thus *f(x)* has exactly one preimage.

Bit-commitment from a pseudo-random generator

Note that since we do not know how to construct a one-way permutation from any one-way function, this section reduces the strength of the cryptographic assumption necessary to construct a bit-commitment protocol.

In 1991 Moni Naor showed how to create a bit-commitment scheme from a cryptographically secure pseudorandom number generator. The construction is as follows. If *G* is pseudo-random generator such that *G* takes *n* bits to *3n* bits, then if Alice wants to commit to a bit *b*

- Bob selects a random *3n*-bit vector *R* and sends *R* to Alice.
- Alice selects a random *n*-bit vector *Y* and computes the *3n*-bit vector *G(Y)*.
- If *b*=1 Alice sends *G(Y)* to Bob, otherwise she sends the bitwise exclusive-or of *G(Y)* and *R* to Bob.

To decommit Alice sends *Y* to Bob, who can then check whether he initially received *G(Y)* or $G(Y) \oplus R$.

This scheme is statistically binding, meaning that even if Alice is computationally unbounded she cannot cheat with probability greater than 2^{-n}. For Alice to cheat, she would need to find a *Y′*, such that $G(Y') = G(Y) \oplus R$. If she could find such a value, she could decommit by sending the truth and Y, or send the opposite answer and Y'. However, G(Y) and G(Y') are only able to produce 2^n possible values while R and the commitment are both picked out of 2^{3n} values in the set of possible *3n*-bit values. She does not pick R, so there is a strong probability that a Y' satisfying the equation required to cheat will not exist.

The concealing property follows from a standard reduction, if Bob can tell whether Alice committed to a zero or one, he can also distinguish the output of the pseudo-random generator *G* from true-random, which contradicts the cryptographic security of *G*.

A perfectly binding scheme based on the discrete log problem

Alice chooses a group of prime order *p*, with generator *g*.

Alice randomly picks a secret value *x* from *0* to $p - 1$ to commit to and calculates $c = g^x$ and publishes *c*. The discrete logarithm problem dictates that from *c*, it is computationally infeasible to compute *x*, so under this assumption, Bob cannot compute *x*. On the other hand, Alice cannot compute a $x' <> x$, such that $g^{x'} = c$, so the scheme is binding.

This scheme isn't perfectly concealing as someone could find the commitment if he manages to solve the discrete logarithm problem.

In fact, this scheme isn't hiding at all with respect to the standard hiding game, where an adversary should be unable to guess which of two messages he chose were committed to - similar to the IND-CPA game. A better example of a perfectly binding commitment scheme is one where the commitment is the encryption of *x* under a semantically secure, public-key encryption scheme, and the decommitment is the string of random bits used to encrypt *x*. An example of an information-theoretically hiding commitment scheme, is the Pedersen commitment scheme, which is binding under the discrete logarithm assumption.

Additionally to the scheme above, it use another generator h of the prime group and a random number r. The commitment is set $c = g^x h^r$.

Quantum bit commitment

It is an interesting question in quantum cryptography if there exist *unconditionally secure* bit commitment protocols on the quantum level, that is protocols which are (at least asymptotically) binding and concealing even if there are no restrictions on the computational resources. One could hope that there might be a way to exploit the intrinsic properties of quantum mechanics, as in the protocols for unconditionally secure key distribution.

However, Dominic Mayers showed in 1996, that this is impossible. Any such protocol can be reduced to a protocol where the system is in one of two pure states after the commitment phase, depending on the bit Alice wants to commit. If the protocol is unconditionally concealing, then Alice can unitarily transform these states into each other using the properties of the Schmidt decomposition, effectively defeating the bindingness property.

One subtle assumption of the proof is that the commit phase must be finished at some point in time. This leaves room for protocols that require a continuing information flow until the bit is unveiled or the protocol is cancelled, in which case it is not binding anymore.

Chapter 14

Digital Signature

A **digital signature** or **digital signature scheme** is a mathematical scheme for demonstrating the authenticity of a digital message or document. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, and that it was not altered in transit. Digital signatures are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering.

Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature, but not all electronic signatures use digital signatures. In some countries, including the United States, India, and members of the European Union, electronic signatures have legal significance. However, laws concerning electronic signatures do not always make clear whether they are digital cryptographic signatures in the sense used here, leaving the legal definition, and so their importance, somewhat confused.

Digital signatures employ a type of asymmetric cryptography. For messages sent through a nonsecure channel, a properly implemented digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital signatures are equivalent to traditional handwritten signatures in many respects; properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes in the sense used here are cryptographically based, and must be implemented properly to be effective. Digital signatures can also provide non-repudiation, meaning that the signer cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid nonetheless. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

Definition

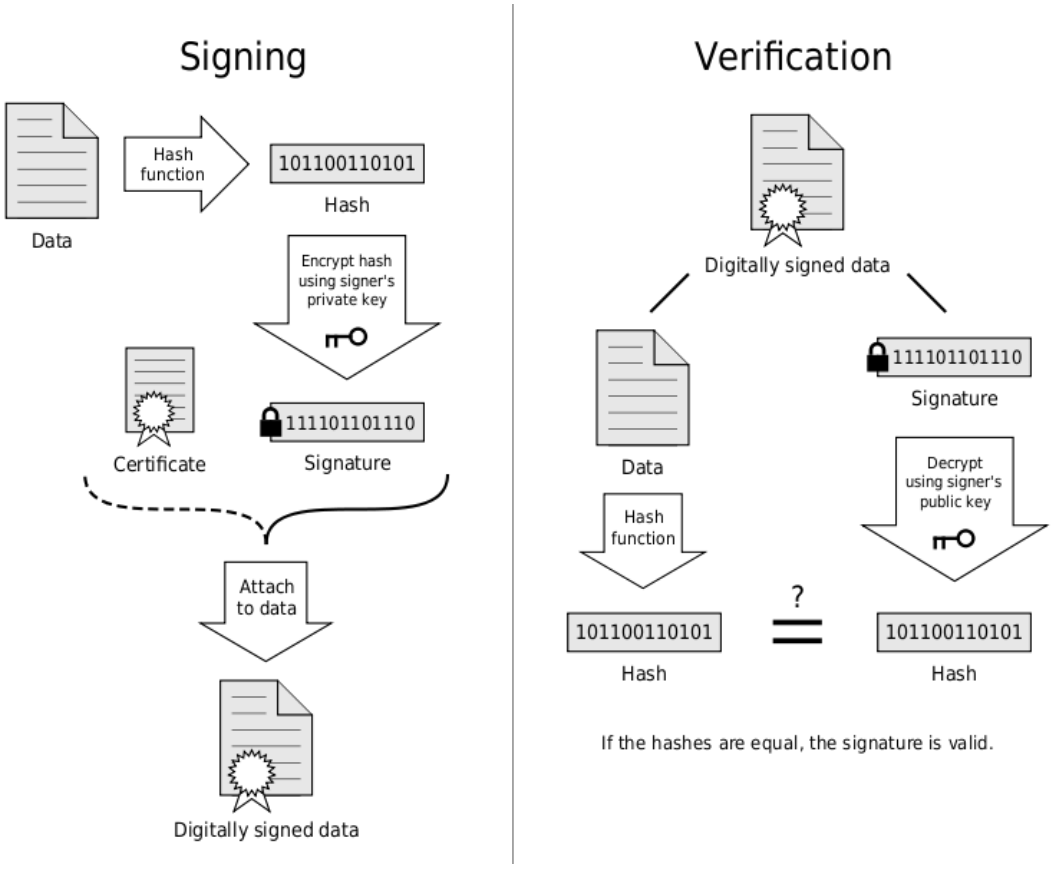


Diagram showing how a simple digital signature is applied and then verified

A digital signature scheme typically consists of three algorithms:

- A *key generation* algorithm that selects a *private key* uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding *public key*.
- A *signing* algorithm that, given a message and a private key, produces a signature.
- A *signature verifying* algorithm that, given a message, public key and a signature, either accepts or rejects the message's claim to authenticity.

Two main properties are required. First, a signature generated from a fixed message and fixed private key should verify the authenticity of that message by using the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party who does not possess the private key.

History

In 1976, Whitfield Diffie and Martin Hellman first described the notion of a digital signature scheme, although they only conjectured that such schemes existed. Soon

afterwards, Ronald Rivest, Adi Shamir, and Len Adleman invented the RSA algorithm, which could be used to produce primitive digital signatures (although only as a proof-of-concept—"plain" RSA signatures are not secure). The first widely marketed software package to offer digital signature was Lotus Notes 1.0, released in 1989, which used the RSA algorithm.

To create RSA signature keys, generate an RSA key pair containing a modulus N that is the product of two large primes, along with integers e and d such that $e\,d \equiv 1 \pmod{\varphi(N)}$, where φ is the Euler phi-function. The signer's public key consists of N and e, and the signer's secret key contains d.

To sign a message m, the signer computes $\sigma \equiv m^d \pmod{N}$. To verify, the receiver checks that $\sigma^e \equiv m \pmod{N}$.

As noted earlier, this basic scheme is not very secure. To prevent attacks, one can first apply a cryptographic hash function to the message m and then apply the RSA algorithm described above to the result. This approach can be proven secure in the so-called random oracle model.

Other digital signature schemes were soon developed after RSA, the earliest being Lamport signatures, Merkle signatures (also known as "Merkle trees" or simply "Hash trees"), and Rabin signatures.

In 1988, Shafi Goldwasser, Silvio Micali, and Ronald Rivest became the first to rigorously define the security requirements of digital signature schemes. They described a hierarchy of attack models for signature schemes, and also present the GMR signature scheme, the first that can be proven to prevent even an existential forgery against a chosen message attack.

Most early signature schemes were of a similar type: they involve the use of a trapdoor permutation, such as the RSA function, or in the case of the Rabin signature scheme, computing square modulo composite n. A trapdoor permutation family is a family of permutations, specified by a parameter, that is easy to compute in the forward direction, but is difficult to compute in the reverse direction without already knowing the private key. However, for every parameter there is a "trapdoor" (private key) which when known, easily decrypts the message. Trapdoor permutations can be viewed as public-key encryption systems, where the parameter is the public key and the trapdoor is the secret key, and where encrypting corresponds to computing the forward direction of the permutation, while decrypting corresponds to the reverse direction. Trapdoor permutations can also be viewed as digital signature schemes, where computing the reverse direction with the secret key is thought of as signing, and computing the forward direction is done to verify signatures. Because of this correspondence, digital signatures are often described as based on public-key cryptosystems, where signing is equivalent to decryption and verification is equivalent to encryption, but this is not the only way digital signatures are computed.

Used directly, this type of signature scheme is vulnerable to a key-only existential forgery attack. To create a forgery, the attacker picks a random signature σ and uses the verification procedure to determine the message *m* corresponding to that signature. In practice, however, this type of signature is not used directly, but rather, the message to be signed is first hashed to produce a short digest that is then signed. This forgery attack, then, only produces the hash function output that corresponds to σ, but not a message that leads to that value, which does not lead to an attack. In the random oracle model, this hash-and-decrypt form of signature is existentially unforgeable, even against a chosen-message attack.

There are several reasons to sign such a hash (or message digest) instead of the whole document.

- **For efficiency:** The signature will be much shorter and thus save time since hashing is generally much faster than signing in practice.
- **For compatibility:** Messages are typically bit strings, but some signature schemes operate on other domains (such as, in the case of RSA, numbers modulo a composite number *N*). A hash function can be used to convert an arbitrary input into the proper format.
- **For integrity:** Without the hash function, the text "to be signed" may have to be split (separated) in blocks small enough for the signature scheme to act on them directly. However, the receiver of the signed blocks is not able to recognize if all the blocks are present and in the appropriate order.

Notions of security

In their foundational paper, Goldwasser, Micali, and Rivest lay out a hierarchy of attack models against digital signatures:

1. In a *key-only* attack, the attacker is only given the public verification key.
2. In a *known message* attack, the attacker is given valid signatures for a variety of messages known by the attacker but not chosen by the attacker.
3. In an *adaptive chosen message* attack, the attacker first learns signatures on arbitrary messages of the attacker's choice.

They also describe a hierarchy of attack results:

1. A *total break* results in the recovery of the signing key.
2. A universal forgery attack results in the ability to forge signatures for any message.
3. A selective forgery attack results in a signature on a message of the adversary's choice.
4. An existential forgery merely results in some valid message/signature pair not already known to the adversary.

The strongest notion of security, therefore, is security against existential forgery under an adaptive chosen message attack.

Uses of digital signatures

As organizations move away from paper documents with ink signatures or authenticity stamps, digital signatures can provide added assurances of the evidence to provenance, identity, and status of an electronic document as well as acknowledging informed consent and approval by a signatory. The United States Government Printing Office (GPO) publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures. Universities including Penn State, University of Chicago, and Stanford are publishing electronic student transcripts with digital signatures.

Below are some common reasons for applying a digital signature to communications:

Authentication

Although messages may often include information about the entity sending a message, that information may not be accurate. Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. The importance of high confidence in sender authenticity is especially obvious in a financial context. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.

Integrity

In many scenarios, the sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to *change* an encrypted message without understanding it. (Some encryption algorithms, known as nonmalleable ones, prevent this, but others do not.) However, if a message is digitally signed, any change in the message after signature will invalidate the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions.

Non-repudiation

Non-repudiation, or more specifically *non-repudiation of origin*, is an important aspect of digital signatures. By this property an entity that has signed some information cannot at a later time deny having signed it. Similarly, access to the public key only does not enable a fraudulent party to fake a valid signature. This is in contrast to symmetric systems,

where both sender and receiver share the same secret key, and thus in a dispute a third party cannot determine which entity was the true source of the information.

Additional security precautions

Putting the private key on a smart card

All public key / private key cryptosystems depend entirely on keeping the private key secret. A private key can be stored on a user's computer, and protected by a local password, but this has two disadvantages:

- the user can only sign documents on that particular computer
- the security of the private key depends entirely on the security of the computer

A more secure alternative is to store the private key on a smart card. Many smart cards are designed to be tamper-resistant (although some designs have been broken, notably by Ross Anderson and his students). In a typical digital signature implementation, the hash calculated from the document is sent to the smart card, whose CPU encrypts the hash using the stored private key of the user, and then returns the encrypted hash. Typically, a user must activate his smart card by entering a personal identification number or PIN code (thus providing two-factor authentication). It can be arranged that the private key never leaves the smart card, although this is not always implemented. If the smart card is stolen, the thief will still need the PIN code to generate a digital signature. This reduces the security of the scheme to that of the PIN system, although it still requires an attacker to possess the card. A mitigating factor is that private keys, if generated and stored on smart cards, are usually regarded as difficult to copy, and are assumed to exist in exactly one copy. Thus, the loss of the smart card may be detected by the owner and the corresponding certificate can be immediately revoked. Private keys that are protected by software only may be easier to copy, and such compromises are far more difficult to detect.

Using smart card readers with a separate keyboard

Entering a PIN code to activate the smart card commonly requires a numeric keypad. Some card readers have their own numeric keypad. This is safer than using a card reader integrated into a PC, and then entering the PIN using that computer's keyboard. Readers with a numeric keypad are meant to circumvent the eavesdropping threat where the computer might be running a keystroke logger, potentially compromising the PIN code. Specialized card readers are also less vulnerable to tampering with their software or hardware and are often EAL3 certified.

Other smart card designs

Smart card design is an active field, and there are smart card schemes which are intended to avoid these particular problems, though so far with little security proofs.

Using digital signatures only with trusted applications

One of the main differences between a digital signature and a written signature is that the user does not "see" what he signs. The user application presents a hash code to be encrypted by the digital signing algorithm using the private key. An attacker who gains control of the user's PC can possibly replace the user application with a foreign substitute, in effect replacing the user's own communications with those of the attacker. This could allow a malicious application to trick a user into signing any document by displaying the user's original on-screen, but presenting the attacker's own documents to the signing application.

To protect against this scenario, an authentication system can be set up between the user's application (word processor, email client, etc.) and the signing application. The general idea is to provide some means for both the user app and signing app to verify each other's integrity. For example, the signing application may require all requests to come from digitally-signed binaries.

WYSIWYS

Technically speaking, a digital signature applies to a string of bits, whereas humans and applications "believe" that they sign the semantic interpretation of those bits. In order to be semantically interpreted the bit string must be transformed into a form that is meaningful for humans and applications, and this is done through a combination of hardware and software based processes on a computer system. The problem is that the semantic interpretation of bits can change as a function of the processes used to transform the bits into semantic content. It is relatively easy to change the interpretation of a digital document by implementing changes on the computer system where the document is being processed. From a semantic perspective this creates uncertainty about what exactly has been signed. WYSIWYS (What You See Is What You Sign) means that the semantic interpretation of a signed message cannot be changed. In particular this also means that a message cannot contain hidden info that the signer is unaware of, and that can be revealed after the signature has been applied. WYSIWYS is a desirable property of digital signatures that is difficult to guarantee because of the increasing complexity of modern computer systems.

Digital signatures vs. ink on paper signatures

An ink signature can be easily replicated from one document to another by copying the image manually or digitally. Digital signatures cryptographically bind an electronic identity to an electronic document and the digital signature cannot be copied to another document. Paper contracts often have the ink signature block on the last page, and the previous pages may be replaced after a signature is applied. Digital signatures can be applied to an entire document, such that the digital signature on the last page will indicate tampering if any data on any of the pages have been altered.

Some digital signature algorithms

- RSA-based signature schemes, such as RSA-PSS
- DSA and its elliptic curve variant ECDSA
- ElGamal signature scheme as the predecessor to DSA, and variants Schnorr signature and Pointcheval-Stern signature algorithm
- Rabin signature algorithm
- Pairing-based schemes such as BLS
- Undeniable signatures
- Aggregate signature - a signature scheme that supports aggregation: Given n signatures on n messages from n users, it is possible to aggregate all these signatures into a single signature whose size is constant in the number of users. This single signature will convince the verifier that the n users did indeed sign the n original messages.

The current state of use — legal and practical

Digital signature schemes share basic prerequisites that— regardless of cryptographic theory or legal provision— they need to have meaning:

1. Quality algorithms
   Some public-key algorithms are known to be insecure, practicable attacks against them having been discovered.
2. Quality implementations
   An implementation of a good algorithm (or protocol) with mistake(s) will not work.
3. The private key must remain private
   if it becomes known to any other party, that party can produce *perfect* digital signatures of anything whatsoever.
4. The public key owner must be verifiable
   A public key associated with Bob actually came from Bob. This is commonly done using a public key infrastructure and the public key↔user association is attested by the operator of the PKI (called a certificate authority). For 'open' PKIs in which anyone can request such an attestation (universally embodied in a cryptographically protected identity certificate), the possibility of mistaken attestation is non trivial. Commercial PKI operators have suffered several publicly known problems. Such mistakes could lead to falsely signed, and thus wrongly attributed, documents. 'closed' PKI systems are more expensive, but less easily subverted in this way.
5. Users (and their software) must carry out the signature protocol properly.

Only if all of these conditions are met will a digital signature actually be any evidence of who sent the message, and therefore of their assent to its contents. Legal enactment cannot change this reality of the existing engineering possibilities, though some such have not reflected this actuality.

Legislatures, being importuned by businesses expecting to profit from operating a PKI, or by the technological avant-garde advocating new solutions to old problems, have enacted statutes and/or regulations in many jurisdictions authorizing, endorsing, encouraging, or permitting digital signatures and providing for (or limiting) their legal effect. The first appears to have been in Utah in the United States, followed closely by the states Massachusetts and California. Other countries have also passed statutes or issued regulations in this area as well and the UN has had an active model law project for some time. These enactments (or proposed enactments) vary from place to place, have typically embodied expectations at variance (optimistically or pessimistically) with the state of the underlying cryptographic engineering, and have had the net effect of confusing potential users and specifiers, nearly all of whom are not cryptographically knowledgeable. Adoption of technical standards for digital signatures have lagged behind much of the legislation, delaying a more or less unified engineering position on interoperability, algorithm choice, key lengths, and so on what the engineering is attempting to provide.

Industry standards

Some industries have established common interoperabiltity standards for the use of digital signatures between members of the industry and with regulators. These include the Automotive Network Exchange for the automobile industry and the SAFE-BioPharma Association for the healthcare industry.

Using separate key pairs for signing and encryption

In several countries, a digital signature has a status somewhat like that of a traditional pen and paper signature, like in the EU digital signature legislation. Generally, these provisions mean that anything digitally signed legally binds the signer of the document to the terms therein. For that reason, it is often thought best to use separate key pairs for encrypting and signing. Using the encryption key pair, a person can engage in an encrypted conversation (*e.g.*, regarding a real estate transaction), but the encryption does not legally sign every message he sends. Only when both parties come to an agreement do they sign a contract with their signing keys, and only then are they legally bound by the terms of a specific document. After signing, the document can be sent over the encrypted link. If a signing key is lost or compromised, it can be revoked to mitigate any future transactions. If an encryption key is lost, a backup or key escrow should be utilized to continue viewing encrypted content. Signing keys should never be backed up or escrowed.

Chapter 15

Secure Communication

When two entities are communicating with each other, and they do not want a third party to listen to their communication, then they want to pass on their message in such a way that no body else could understand their message. This is known as communicating in a secure manner or **secure communication**. Secure communication includes means by which people can share information with varying degrees of certainty that third parties cannot know what was said. Other than communication spoken face to face out of possibility of listening, it is probably safe to say that no communication is guaranteed secure in this sense, although practical limitations such as legislation, resources, technical issues (interception and encryption), and the sheer volume of communication are limiting factors to surveillance.

The purpose here is to describe the various means by which security is sought and compromised, the differing kinds of security possible, and the current means and their degree of security readily available.

With many communications taking place over long distance and mediated by technology, and increasing awareness of the importance of interception issues, technology and its compromise are at the heart of this debate. For this reason, here we focusses on communications mediated or intercepted by technology.

Also see *Trusted Computing*, an approach under present development that achieves security in general at the potential cost of compelling obligatory trust in corporate and governmental bodies.

Users and needs

Many forms of everyday communication are "reasonably" secure, thus, we do not assume telephone calls are intercepted when we use them. However in some areas such as online intellectual property rights, legal, criminal, political and commercial communications, this assumption is inadequate.

History

In 1898, Nikola Tesla demonstrated a radio controlled boat in Madison Square Garden that allowed secure communication between transmitter and receiver.

One of the most famous forms of secure communication was the Green Hornet. During WWII, Winston Churchill had to make vital calls to the President of the United States, Franklin D. Roosevelt. These calls talked about such things as shipping and troop movements. At first, the calls were made using a radio phone as this was thought to be secure. Unfortunately, due to the Nazis having a listening station in Holland they were able to hear every last word. As soon as it was realized they stopped using the radio phone and started work on a whole new system, the Green Hornet. This meant that anyone listening in would just hear white noise but as the only two identical copies were held with the Prime Minister and the President the conversation was clear to them. As secrecy was paramount, the location of the Green Hornet was only known by the people who built it and Winston Churchill, and if anyone did see him entering the room it was kept in, all they would see was the Prime Minister entering a closet labeled 'Broom Cupboard.' It is the said that because of the way the Green Hornet works it is not able to be beaten, even today.

Nature and limitations of secure communication

Types of security

Security can be broadly categorised under the following headings, with examples:

- Hiding the content or nature of a communication
    - Code

A code is a rule for converting a piece of information (for example, a letter, word, phrase, or gesture) into another form or representation (one sign into another sign), not necessarily of the same type. In communications and information processing, encoding is the process by which information from a source is converted into symbols to be communicated. Decoding is the reverse process, converting these code symbols back into information understandable by a receiver. One reason for coding is to enable communication in places where ordinary spoken or written language is difficult or impossible. For example, semaphore, where the configuration of flags held by a signaller or the arms of a semaphore tower encodes parts of the message, typically individual letters and numbers. Another person standing a great distance away can interpret the flags and reproduce the words sent.

    - Encryption
    - Steganography
    - Identity Based

- Hiding the parties to a communication (prevention of identification, or anonymity)
    - "Crowds" and similar anonymous group structures. i.e. it is difficult to identify who said what when it comes from a "crowd".
    - Anonymous communication devices (unregistered cellphones, Internet cafes)
    - Anonymous proxies
    - Hard to trace routing methods (through unauthorized 3rd party systems, or relays)
- Hiding the fact that a communication takes place
    - "Security by obscurity" (similar to needle in a haystack)
    - Random traffic (creating random data flow in order that the presence of genuine communication is harder to detect and traffic analysis less reliable)

Each of the three is important, and depending on the circumstances any of these may be critical. For example, if a communication is not readily identifiable, then it is unlikely to attract attention for identification of parties, and the mere fact a communication has taken place (regardless of content) is often enough by itself to establish an evidential link in legal prosecutions. It is also important with computers, to be sure where the security is applied, and what is covered.

Borderline cases

A further category, which touches upon secure communication, is software intended to take advantage of security openings at the end-points. This software category includes trojan horses, keyloggers and other spyware.

These types of activity are usually addressed with everyday mainstream security methods, such as antivirus software, firewalls, programs that identify or neutralize adware and spyware, as well as web filtering programs such as proxomitron and privoxy which check all web pages being read and identify and remove common nuisances contained. As a rule they fall under computer security rather than secure communications.

Tools used to obtain security

Encryption

Encryption is where data is rendered hard to read by an unauthorised party. Since encryption can be made extremely hard to break, many communication methods either use deliberately weaker encryption than possible, or have backdoors inserted to permit rapid decryption. In some cases government authorities have required backdoors be installed in secret. Many methods of encryption are also subject to "man in the middle" attack whereby a third party who can 'see' the establishment of the secure communication is made privy to the encryption method, this would apply for example to interception of

computer use at an ISP. Provided it is correctly programmed, sufficiently powerful, and the keys not intercepted, encryption would usually be considered secure.

The encryption can be implemented in a way to require the use of encryption, i.e. if encrypted communication is impossible then no traffic is sent, or opportunistically. Opportunistic encryption is a lower security method to generally increase the percentage of generic traffic which is encrypted. This is analogous to beginning every conversation with "Do you speak Navajo?" If the response is affirmative, then the conversation proceeds in Navajo, otherwise it uses the common language of the two speakers. This method does not generally provide authentication or anonymity but it does protect the content of the conversation from eavesdropping.

Steganography

Steganography ("hidden writing") is the means by which data can be hidden within other more innocuous data. Thus a watermark proving ownership embedded in the data of a picture, in such a way it is hard to find or remove unless you know how to find it. or, for communication, the hiding of important data (such as a telephone number) in apparently innocuous data (an MP3 music file). An advantage of steganography is plausible deniability, that is, unless one can prove the data is there (which is usually not easy), it is deniable that the file contains any.

Identity based networks

Unwanted or malicious behavior is possible on the web since it is inherently anonymous. True identity based networks replace the ability to remain anonymous and are inherently more trustworthy since the identity of the sender and recipient are known. (The telephone system is an example of an identity based network.)

Anonymized networks

Recently, anonymous networking has been used to secure communications. In principle, a large number of users running the same system, can have communications routed between them in such a way that it is very hard to detect what any complete message is, which user sent it, and where it is ultimately going from or to. Examples are Crowds, Tor, I2P, Mixminion, various anonymous P2P networks, and others.

Anonymous communication devices

In theory, an unknown device would not be noticed, since so many other devices are in use. This is not altogether the case in reality, due to the presence of systems such as Carnivore and Echelon which can monitor communications over entire networks, and the fact that the far end may be monitored as before. Examples include payphones, Internet cafe, etc.

Methods used to "break" security

Bugging

The placing covertly of monitoring and/or transmission devices either within the communication device, or in the premises concerned.

Computers (general)

Any security obtained from a computer is limited by the many ways it can be compromised - by hacking, keystroke logging, backdoors, or even in extreme cases by monitoring the tiny electrical signals given off by keyboard or monitors to reconstruct what is typed or seen (TEMPEST, which is quite complex).

Laser reading of windows

In certain cases individuals have had private spoken communications intercepted by means of laser. This usually involves a sensitive laser directed at a window, capable of picking up the tiny glass movements caused by sounds, and conversion back to speech.

Systems offering a degree of secure communication

Anonymous cellphones

Cellphones can easily be obtained, but are also easily traced and "tapped". There is no (or only limited) encryption, the phones are traceable - often even when switched off - since the phone and SIM card broadcast their International Mobile Subscriber Identity (IMSI). It is possible for a cellphone company to turn on some cellphones when the user is unaware and use the microphone to listen in on you, and according to James Atkinson, a counter-surveillance specialist cited in the same source, "Security-conscious corporate executives routinely remove the batteries from their cell phones" since many phones' software can be used "as-is", or modified, to enable transmission without user awareness and the user can be located within a small distance using signal triangulation and now using built in GPS features for newer models.

Landlines

Analogue landlines are not encrypted and are trivially tapped. Such tapping requires physical access to the line which is easily obtained from a number of places, e.g. distribution points, cabinets and the exchange itself. Tapping a landline in this way would also enable the attacker to make calls which appear to originate from the tapped line.

Anonymous Internet

Using a third party system of any kind (payphone, Internet cafe) is often quite secure, however if that system is used to access known locations (a known email account or 3rd

party) then it may be tapped at the far end, or noted, and this will remove any security benefit obtained. Some countries also impose mandatory registration of Internet cafe users.

Anomymous proxies are another common type of protection, which allow one to access the net via a third party (often in a different country) and make tracing difficult. Note that there is seldom any guarantee that the plaintext is not tappable, nor that the proxy does not keep its own records of users or entire dialogs. As a result anonymous proxies are a generally useful tool but may not be as secure as other systems whose security can be better assured. Their most common use is to prevent a record of the originating IP, or address, being left on the target site's own records. Typical anonymous proxies are found at both regular websites such as Anonymizer.com and spynot.com, as well as on proxy sites which maintain up to date lists of large numbers of temporary proxies in operation.

A recent development on this theme arises when wireless Internet connections ("Wi-fi") are left in their unsecured state. The effect of this is that any person in range of the base unit can piggyback the connection - that is, use it without the owner being aware. Since many connections are left open in this manner, situations where piggybacking might arise (willful or unaware) have successfully led to a defense in some cases, since it makes it difficult to prove the owner of the connection was the downloader, or had knowledge of the use to which unknown others might be putting their connection. An example of this was the Tammie Marson case, where neighbours and anyone else might have been the culprit in the sharing of copyright files. Conversely, in other cases, people deliberately seek out businesses and households with unsecured connections, for illicit and anonymous Internet usage, or simply to obtain free bandwidth.

Programs offering more secure communications

- Skype - secure voice over Internet, secure chat. Uses 128-bit AES (256-bit is the standard) and 1024-bit asymmetrical protocols to exchange initial keys (which is considered relatively weak by NIST). Proprietary. No information on backdoors. An article in 2004 suggested that Skype has relatively weak encryption, but more recent analyses, one by invitation and one by reverse engineering presented at DEF CON 2005, both conclude that Skype uses encryption effectively. Criticism focuses upon its proprietary "black box" design, its relatively short (1536 bit) keys, excessive bandwidth use of user supernodes, and excessive trust of other computers able to "speak Skype".
- Zfone is an open source secure voice over Internet program, by Phil Zimmermann, the creator of PGP.
- I2P-Messenger is a simple secure (end-to-end encrypted), anonymous, and serverless instant messenger with file transfer support.
- pbxnsip is a SIP-based PBX that uses TLS and SRTP for encrypting the voice traffic. In contrast to other proprietary protocols, the protocol is open so that devices from independent vendors can be used. The encryption includes the relay of Instant Messaging and Presence information as well as the management interface.

- Secure IRC and web chat - Some IRC clients and systems use security such as SSL. This is not standardised. Likewise some web chat clients such as Yahoo Messenger use secure communications on their web based program. Again the security of these is unverified, and it is likely the communication is not secured other than to and from the client.
- Trillian - offers secure IM facility, however appears to have weaknesses in key exchange which would enable a "man in the middle" attack with ease. Proprietary, no information on backdoors.
- Off-the-Record Messaging is a plugin which adds end-to-end encryption and authentication as well as Perfect forward secrecy to instant messaging. It is not a separate protocol but runs under most every IM protocol.
- WASTE - open source secure IM, high strength "end to end" encryption, within an anonymised network.
- Secure email - some email networks such as "hushmail" or Opolis Secure Mail, are designed to provide encrypted and/or anonymous communication. They authenticate and encrypt on the users own computer, to prevent transmission of plain text, and mask the sender and recipient. Mixminion and I2P-Bote provide a higher level of anonymity by using a network of anonymizing intermediaries, (similar to how Tor and crowds work above, but at a higher latency).
- AESpad.com - open source online encrypted secure chat. Uses 256-bit AES symmetrical encryption. Relies on a pre-shared key between chat participants.
- ChatCrypt.com - another online encrypted secure chat. It uses the 256-bit AES symmetrical encryption in CTR mode.

Chapter 16

Disk Encryption Theory

Disk encryption is a special case of *data at rest* protection when the storage media is a sector-addressable device (e.g., a hard disk). Here we, presents cryptographic aspects of the problem. For discussion of different software packages and hardware devices devoted to this problem see disk encryption software and disk encryption hardware.

Problem definition

Disk encryption methods aim to provide three distinct properties:

1. The data on the disk should remain confidential
2. Data retrieval and storage should both be fast operations, no matter where on the disk the data is stored.
3. The encryption method should not waste disk space (i.e., the amount of storage used for encrypted data should not be significantly larger than the size of plaintext)

The first property requires defining an adversary with respect to whom the data is being kept confidential. The strongest adversaries studied in the field of disk encryption have these abilities:

1. they can read the raw contents of the disk at any time;
2. they can request the disk to encrypt and store arbitrary files of their choosing;
3. and they can modify unused sectors on the disk and then request their decryption.

A method provides good confidentiality if the only information such an adversary can determine over time is whether the data in a sector has or has not changed since the last time they looked.

The second property requires dividing the disk into several *sectors*, usually 512 bytes (4096 bits) long, which are encrypted and decrypted independently of each other. In turn, if the data is to stay confidential, the encryption method must be *tweakable* – no two sectors should be processed in exactly the same way. Otherwise, the adversary could

decrypt any sector of the disk by copying it to an unused sector of the disk and requesting its decryption.

The third property is generally noncontroversial. However, it indirectly prohibits the use of stream ciphers, since stream ciphers require, for their security, that the same initial state not be used twice (which would be the case if a sector is updated with different data); thus this would require an encryption method to store separate initial states for every sector on disk – seemingly a clear waste of space. The alternative – a block cipher – is limited to a certain block size (usually 128 or 256 bits). Because of this, disk encryption chiefly studies chaining modes, which expand the encryption block length to cover a whole disk sector. The considerations already listed make several well-known chaining modes unsuitable: ECB mode, which cannot be tweaked, and modes that turn block ciphers into stream ciphers, such as the CTR mode.

These three properties do not provide any assurance of disk integrity – that is, they don't tell you whether an adversary has been modifying your ciphertext. In part, this is because an absolute assurance of disk integrity is impossible: no matter what, an adversary could always revert the entire disk to a prior state, circumventing any such checks. If some non-absolute level of disk integrity is desired, it can be achieved within the encrypted disk on a file-by-file basis using message authentication codes.

Block cipher-based modes

All block cipher-based methods make use of so-called *modes*, which allow encrypting larger amounts of data than the ciphers' block-size (typically 128 bits). Modes are therefore rules on how to repeatedly apply the ciphers' single-block operations.

CBC

Cipher block chaining (CBC) is a common chaining mode in which the previous block's ciphertext is xored with the current block's plaintext before encryption:

$$C_i = E_K(C_{i-1} \oplus P_i)$$

Since there isn't a "previous block's ciphertext" for the first block, an initialization vector (IV) must be used as C_{-1}. This, in turn, makes CBC tweakable in some ways.

CBC suffers from some problems. For example, ***if*** the IVs are predictable an adversary can manage to store a file specially created to zero out the IV, it is possible to leave a "watermark" on the disk, proving that the specially created file is, indeed, stored on disk. The exact method of constructing the watermark depends on the exact function providing the IVs; but the general recipe is to create two encrypted sectors which have identical first blocks b_1 and b_2; these two are then related to each other by $b_1 \oplus IV_1 = b_2 \oplus IV_2$. Thus, when these two blocks are encrypted, they both encrypt to the same thing, leaving a watermark on the disk. The exact pattern of "same-different-same-different" on disk can then be altered to make the watermark unique to a given file.

To protect against the watermarking attack, a cipher or a hash function is used to generate the IVs from the key and the current sector number, so that an adversary cannot predict them. In particular, the ESSIV approach discussed below uses a block cipher in CTR mode to generate the IVs.

LRW

In order to prevent such elaborate attacks, different modes of operation were introduced: tweakable narrow-block encryption (LRW and XEX) and wide-block encryption (CMC and EME).

Whereas a purpose of a usual block cipher E_K is to mimic a random permutation for any secret key K, the purpose of *tweakable* encryption E_K^T is to mimic a random permutation for any secret key K and any known tweak T. The tweakable narrow-block encryption (LRW) is an instantiation of the mode of operations introduced by Liskov, Rivest, and Wagner. This mode uses two keys: K is the key for the block cipher and F is an additional key of the same size as block. For example, for AES with a 256-bit key, K is a 256-bit number and F is a 128-bit number. Encrypting block P with logical index (tweak) I uses the following formula: $E_K(P \oplus X) \oplus X$, where $X = F \otimes I$. Here multiplication $\otimes$ and addition $\oplus$ are performed in the finite field (GF(2^{128}) for AES). With some precomputation, only a single multiplication per sector is required (note that addition in a binary finite field is a simple bitwise addition, also known as xor): $F \otimes I = F \otimes (I_0 \oplus \delta) = F \otimes I_0 \oplus F \otimes \delta$, where $F \otimes \delta$ are precomputed for all possible values of δ. This mode of operation needs only a single encryption per block and protects against all the above attacks except a minor leak: if the user changes a single plaintext block in a sector then only a single ciphertext block changes. (Note that this is not the same leak the ECB mode has: with LRW mode equal plaintexts in different positions are encrypted to different ciphertexts.)

Some security concerns exist with LRW, and this mode of operation has now been replaced by XTS.

LRW is employed by Bestcrypt and supported as an option for dm-crypt and FreeOTFE disk encryption systems.

XEX

Another tweakable encryption mode XEX (Xor-Encrypt-Xor), was designed by Rogaway to allow efficient processing of consecutive blocks (with respect to the cipher used) within one data unit (e.g. a disk sector). The tweak is represented as a combination of the sector address and index of the block inside the sector (the original XEX mode proposed by Rogaway allows to have several indexes). To encrypt block j in sector I, the following formula is used $C = E_K(P \oplus X) \oplus X$, where $X = E_K(I) \otimes \alpha^j$ and α is the primitive element of GF(2^{128}) defined by polynomial x, e.g. the number "2".

The basic blocks of the LRW mode (AES cipher and Galois field multiplication) are the same as the ones used in the Galois/Counter Mode (GCM) thus permitting a compact implementation of the universal LRW/XEX/GCM hardware.

XTS

XTS is XEX-based Tweaked CodeBook mode (TCB) with CipherText Stealing (CTS). Although XEX-TCB-CTS would more consistently be abbreviated as XTC, “C” was replaced with “S” (for “stealing”) to avoid confusion with the abbreviated *ecstasy*. Ciphertext stealing provides support for sectors with size not divisible by block size, for example, 520-byte sectors and 16-byte blocks. XTS-AES was standardized on 2007-12-19 as IEEE P1619.

On January 27, 2010, NIST released Special Publication (SP) 800-38E in final form. SP 800-38E is a recommendation for the XTS-AES mode of operation, as standardized by IEEE Std 1619-2007, for cryptographic modules. The publication approves the XTS-AES mode of the AES algorithm by reference to the IEEE Std 1619-2007, subject to one additional requirement, which limits the maximum size of each encrypted data unit (typically a sector or disk block) to 2^{20} AES blocks. According to SP 800-38E, "In the absence of authentication or access control, XTS-AES provides more protection than the other approved confidentiality-only modes against unauthorized manipulation of the encrypted data."

As of September 2010, XTS is supported by BestCrypt, dm-crypt, FreeOTFE, TrueCrypt, DiskCryptor, FreeBSD and OpenBSD softraid disk encryption software (also native in Mac OS X Lion's FileVault), in hardware-based media encryption devices by the SPYRUS Hydra PC Digital Attaché and the Kingston DataTraveler 5000.

Issues with XTS

XTS makes use of two different keys, usually generated by splitting the supplied block cipher's key in half, which is the major difference to XEX, without adding any additional security, but complicating the process. According to this source, the reason for this seems to be rooted in a misinterpretation of the original XEX-paper.

CMC and EME

CMC and EME protect even against the minor leak mentioned above. Unfortunately, the price is a twofold degradation of performance: each block must be encrypted twice; many consider this to be too high a cost, since the same leak on a sector level is unavoidable anyway.

CMC, introduced by Halevi and Rogaway, stands for CBC-mask-CBC: the whole sector encrypted in CBC mode (with $C_{-1} = E_A(I)$), the ciphertext is masked by xoring with $2(C'_0 \oplus C'_{k-1})$, and re-encrypted in CBC mode starting from the last block. When the

underlying block cipher is a strong pseudorandom permutation (PRP) then on the sector level the scheme is a tweakable PRP. One problem is that in order to decrypt P_0 one must sequentially pass over all the data twice.

In order to solve this problem, Halevi and Rogaway introduced a parallelizable variant called EME (ECB-mask-ECB). It works in the following way:

- the plaintexts are xored with $L = E_K(0)$, shifted by different amount to the left, and are encrypted: $P'_i = E_K(P_i \oplus 2^i L)$;
- the mask is calculated: $M = M_P \oplus M_C$, where $M_P = I \oplus \bigoplus P'_i$ and $M_C = E_K(M_P)$;
- intermediate ciphertexts are masked: $C'_i = P'_i \oplus 2^i M$ for $i = 1, \ldots, k-1$ and $C'_0 = M_C \oplus I \oplus \bigoplus_{i=1}^{k-1} C'_i$;
- the final ciphertexts are calculated: $C_i = E_K(C'_i) \oplus 2^i L$ for $i = 0, \ldots, k-1$.

Note that unlike LRW and CMC there is only a single key K.

CMC and EME were considered for standardization by SISWG. CMC was rejected for technical considerations. EME is patented, and so is not favored to be a primary supported mode.

ESSIV

Encrypted Salt-Sector Initialization Vector (ESSIV) is a method for generating initialization vectors for block encryption to use in disk encryption. The usual methods for generating IVs are predictable sequences of numbers based on, for example, time stamp or sector number, and permits certain attacks such as a watermarking attack. ESSIV prevents such attacks by generating IVs from a combination of the sector number with the hash of the key. It is the combination with the key in form of a hash that makes the IV unpredictable.

$$IV(\text{sector}) \quad = \quad E_s(\text{sector}), \quad \text{where} \quad s = \text{hash}_K$$

ESSIV was designed by Clemens Fruhwirth and has been integrated into the Linux kernel since version 2.6.10, though a similar scheme has been used to generate IVs for OpenBSD's swap encryption since 2000.

ESSIV is supported as an option by the dm-crypt and FreeOTFE disk encryption systems.