An Introduction to Cryptography

Jon Callas
Chief Technology Officer and Chief Security Officer

Rest Secured™

Release Information

An Introduction to Cryptography, released October 2006.

Copyright Information

Licensing and Patent Information

The IDEA cryptographic cipher described in US patent number 5,214,703 is licensed from Ascom Tech AG. The CAST encryption algorithm is licensed from Northern Telecom, Ltd. PGP Corporation has secured a license to the patent rights contained in the patent application Serial Number 10/655,563 by The Regents of the University of California, entitled Block Cipher Mode of Operation for Constructing a Wide-blocksize block Cipher from a Conventional Block Cipher. PGP Corporation may have patents and/or pending patent applications covering subject matter in this software or its documentation; the furnishing of this software or documentation does not give you any license to these patents.

Trademarks

PGP, the PGP logo, *Pretty Good Privacy*, and *Pretty Good* are all registered trademarks of PGP Corporation. *Rest Secured* is a trademark of PGP Corporation. All other registered and unregistered trademarks are the sole property of their respective owners.

Acknowledgments

The compression code in PGP software is by Mark Adler and Jean-Loup Gailly, used with permission from the free Info-ZIP implementation.

Limitations

The information in this document is subject to change without notice. PGP Corporation does not warrant that the information meets your requirements or that the information is free of errors. The information may include technical inaccuracies or typographical errors. Changes may be made to the information and incorporated in new editions of this document, if and when made available by PGP Corporation.

Export Information

Export of PGP software may be subject to compliance with the rules and regulations promulgated from time to time by the Bureau of Industry and Security, US Department of Commerce, which restrict the export and re-export of certain products and technical data.

About PGP Corporation

PGP Corporation, a global security software company, is the leader in email and data encryption. Based on a unified key management and policy infrastructure, the PGP® Encryption Platform offers the broadest set of integrated applications for enterprise data security. The platform enables

organizations to meet current needs and expand as security requirements evolve for email, laptops, desktops, instant messaging, PDAs, network storage, FTP and bulk data transfers, and backups. PGP solutions are used by more than 30,000 enterprises, businesses, and governments worldwide, including 84 percent of the Fortune® 100 and 66 percent of the Fortune® Global 100, As a result, PGP Corporation has earned a global reputation for innovative, standards-based, and trusted solutions. PGP solutions help protect confidential information, secure customer data, achieve regulatory and audit compliance, and safeguard companies' brands and reputations. Contact PGP Corporation at http://www.pgp.com or +1 650 319 9000.

Contents

Chapter 1

About This Book

The right to privacy is protected under the Constitution in various ways.
– United States Chief Justice John Roberts

Who Should Read This Book

Anyone should read this book who is interested in more than the *what* of the subject, but also the *why* and the *how* as well. This book is a basic explanation of the details, the terminology and technology behind cryptography in general and PGP software in specific. If you want to understand cryptography, this is a good place to start. Furthermore, this book has a lot of links to instructive web sites as well as many more sources to go to if you are interested. If you are reading this book in electronic PDF form, you can click the links in blue and they will take you to the reference.

Bricks Made of Mist

Cryptography is an important part of information technology. It is how we do things that would be straightforward with physical objects when we are working with systems that are nothing but information. If I send you a letter on paper, I can put it in an envelope so no one else can read it. I can sign it at the bottom. It gets a postmark that tells you a bit about where it came from. In business, we might exchange cards or I might show you a customer card that indicates a close relationship between us, such as a frequent customer card. The way we do these things on the Internet, talking privately or demonstrating relationships, is through the use of cryptography. Cryptography brings simple things in the concrete world into the virtual realm. Cryptography is assurance at a distance, privacy at a distance, authenticity at a distance. It is how we get solidity in a world that's made of nothing but ones and zeros.

Cryptography is Hard — And That Makes it Easy

There are a number of reasons why cryptography is hard. It is hard because you already know a good deal of the basics. You probably wrote a secret message when you were a kid. You've probably

at least looked at newspaper cryptograms, if not solved them yourself. Consequently, the basic terms are ones with which you're already familiar. However, cryptography has changed more in the last thirty years than it did in the previous 3,000. This is because modern cryptography is tied up with computers. Also, the most significant changes are ones that have no direct analogue in nature. They have an almost Alice In Wonderland[1] quality to them. Public-key cryptographic systems — the core of PGP software and all modern cryptography — are particularly counterintuitive. Cryptography is also hard because you will have to unlearn a few things that you know, or perhaps a better way to say it is that you'll have to expand what you already know to include some new, subtle ideas as well.

Cryptography is hard because the problems we are solving, the goals we achieve when we build cryptographic systems are easy to state and understand. However, the technical mechanisms we use are hard to build, hard to explain, and hard to understand. The good news, though, is that you're not alone. The best cryptographers in the world are constantly making mistakes, constantly having to go back to the old drawing board. This means you're on a similar footing to those of us who have a lot of experience with cryptography. If something sounds screwy, it probably is.

As I'm writing this book, I'm also reading an email discussion of a new cryptographic system. It's easy to explain – I want to be able to send you a message and want you to know that it hasn't been changed in transit, and the email thread is about mechanism for that. The discussion has degenerated into name-calling: "You haven't proved that." "Yes, I have." "No, you haven't." "I sent a proof to the mailing list." "That's not a proof." "Yes, it is." "No, it isn't, you fool." "It is, too, and I'm not going to stand for being called names." "Yes, you are going to stand for it, because you're only proving me right that you *don't* have a proof and you *are* a fool." Cryptography is so hard that it reduces grown men and women with fancy titles like Professor of Mathematics to parodies of Bugs Bunny and Daffy Duck.

And that's why cryptography is also easy. It's so hard that people may have more experience than you, but they're not any more sure of it. Cryptography is easy because there is no embarrassment in needing something explained to you four times before you get it. The best cryptographers in the world have gotten that way by making the most mistakes. It's so hard that there's room to have unconventional opinions because many things are under active, noisy debate. This ongoing debate makes it not only an exciting technical discipline, but also an enjoyable spectator sport.

Perfectly Hard or Hardly Perfect?

In understanding cryptography, it's also important to understand some of the terms we use and the way we use them. There are, unfortunately, many things we will say that do not have formal, rigorous definitions. As examples, I will talk about *hard* problems or a *perfect* system. Although there's no formal, agreed-upon, mathematical definition of a hard problem, a hard problem is one where there isn't a solution that is better than guessing. A perfect system is one based on a hard problem.

Now, colloquially, what many cryptographers mean by *hard* is that it also be practical, and this distinction is where I'm going to break ground with them. I'm going to talk differently about practicality and hardness because I think there is value in separating them.

[1] We'll hear a lot more about Alice and her other friends as well.

To be ridiculous, let's presume we're going to talk about a *perfect* encryption system, one that is based on truly *hard* mathematics, but there are only three possible solutions. It's not very practical — guess all three options and then you're done. It's idiotically impractical as a way to keep secrets. But the advantage of hard problems is that if you increase the number of options from 3 to 340,282,366,920,938,463,463,374,607,431,768,211,456 (which happens to be 2^{128}), then suddenly it's practical.

What *is* Cryptography, Anyway?

Because you already know a lot of the basics, I'll jump into some definitions. *Cryptography* is the art and science of secret writing. The word itself comes from the Greek for "hidden writing." When we use cryptography, we start with ordinary data that we call *plaintext* and produce from it something unreadable, called *ciphertext*. The recipe that we use for transforming plaintext to ciphertext and back again is a *cipher*. A cipher also uses a secret as part of its transformation, and that secret is called a *key*. Turning plaintext into ciphertext is called *encrypting*, and turning ciphertext into plaintext is called *decrypting*. Thus we can say:

Encrypting: $\boldsymbol{ciphertext = cipher(key, plaintext)}$

and

Decrypting: $\boldsymbol{plaintext = cipher(key, ciphertext)}$

Related to ciphers are systems called *codes*. A code is merely a table that makes a correspondence between symbols (typically numbers) and letters, words, etc. For example, anyone who uses a computer is using a code that numbers various characters. As an example, the letter A is given the number 49 in the code I'm writing in now, called Unicode [Unicode]. In the heyday of telegraphy, codes were used that had numbers for words or entire phrases [Codebooks]. Codes and ciphers were often used together. That is still true with computers today, because all text we use is composed of codes. Ciphers differ from codes in that a cipher has a secret variable called a *key* that (if everything goes according to plan) modifies the encrypted data so it is unreadable by anyone who doesn't know the key. However, if a code is a secret code, that is also a form of cryptography, a form of hidden writing.

Cryptography has a sister discipline called *cryptanalysis*, which is the art and science of breaking the ciphers and codes that a cryptographer creates. Together, cryptography and cryptanalysis form the discipline of *cryptology*, but colloquially, the word cryptography is often used to mean cryptology. I am frequently guilty of this imprecision.

There is another related discipline called *steganography*. Steganography comes from the Greek meaning "covered writing." Cryptography literally means hidden writing, but it's not hidden, it's just not readable. You can see an encrypted message. Steganography is actually hiding messages (which may also be encrypted). For example, a steganographic system might hide a message in a picture or music so that it can't be seen or heard. Invisible inks and hollow heels in shoes are also forms of steganography [STEGO]. Breaking steganographic systems is called *steganalysis*.

An important difference between codes, ciphers, and steganography[2] is that codes and steganographic systems are fixed systems. If you have a code book and therefore know that the number for the word *transparent* is 22611 [Slater], then someone using that code book for secrecy needs to *overhaul* (15740) their code. Similarly, steganography is difficult to use practically because it is hard to have standard hiding places that don't give away where to look. Effective steganography must be custom-built.

In contrast, cryptography has the advantage that the entire system can be made public. The only *secret* part of a properly built cryptographic system are the keys. Consequently, a cryptographic system can be publicly debated, reviewed, and improved without hurting the security of the people who use it.

At PGP Corporation, we rely on this principle in using only standard, reviewed components. We also extend it to you by making our software available for review [PGPsource] so that anyone can look at our software and verify that it is correctly built. Over 630 people per month took us up on this offer in 2006.

A History of This Book

When Phil Zimmermann wrote the first PGP software, it included text files that described the PGP program and its basic operation as well as increasing detail, including the actual data formats of PGP messages. That grew into an entire book, including all of these things [PGP2]. The source for the PGP program was published in another book, the start of PGP's tradition of open access to the software [PGP2S]. O'Reilly published a book in 1995 [ORPGP], that covered a lot of of ground and history.

The original *An Introduction to Cryptography* appeared as a part of PGP 6.0 in 1998. The product manuals for the PGP software described the basic operation of the system. We published the source code for the product in its own set of books. PGP as a protocol had become the OpenPGP standard [OpenPGP] [OPGPMIME]. There were also a number of other technical books that described the details of how the core cryptography works for programmers, engineers, scientists, and mathematicians, but nothing available that covered the subject for intelligent, curious readers to start from and then progress to whatever level of detail they wanted. Thus *An Introduction to Cryptography* was written to serve this need.

Many things have changed since then. The United States' export regulations permit us to put the PGP source code [PGPsource] directly on the web site rather than having to publish it in books[3]. The PGP software has grown from being tightly controlled to one that is available in more than one-hundred countries. Ten years ago, the PGP software was very political; using it was almost an act of defiance. Today, it is pretty ordinary, and often fulfills a business or legal requirement to encrypt data.

[2] If you want to make a rigorous taxonomy, you might rightly term codes as a form of steganography. Historically, codes and ciphers have always gone together but hiding techniques have been considered a separate discipline — breaking codes and ciphers being different than breaking invisible inks, for example. So I'll keep lumping codes and ciphers together and separate from steganography.

[3] The US export regulations said then and still say that printed material, including source code, are exempt from them. This allowed us to legally make the program available for peer review by exporting printed copies. That was an awkward and unwieldy process, and thankfully no longer necessary.

Although there are many other resources available, there is still no replacement for *An Introduction to Cryptography* as a good place to learn about cryptography, the PGP software, and then where to go. This book is still useful, still needed. We've completely revised it to take into account the way the world has changed not only since 1991, but since 1998.

Special Thanks

Paulina Borsook edited, gave aid, support, research, writing assistance, and much-needed snark. Olivia Dillan and Will Price insisted this book had to be re-written. Barbara Jurin provided her usual fine editing. Phil Zimmermann also edited and insisted that the tone of this book be conversational. Tom Stoppard taught me that you can't explain something thoroughly without an infinite series of right parentheses.

What's New and Changed

In October, 2006, we fixed some typos and misspellings, none too horrible, unless it was your name I misspelled.

Chapter 2

Why Cryptography is Important

> *If you reveal your secrets to the wind you should not blame the wind for revealing them to the trees.*
>
> – Kalil Gibran, *Sand and Foam*

People are creatures that communicate. We also choose to whom we tell what. From the earliest age, not only do we talk, but we whisper. Not only do we write, but we pass notes. Shortly after we learn to talk, we learn to whom to say what.

Cryptography is important because on the surface it is about making something secret, but it is also about controlling *access*, specifying who can get to information under what terms. Very likely, you are studying crypto partially out of intellectual curiosity, but also because of some actual *requirement* that you have. As the world's economy becomes more reliant on information rather than physical goods, cryptography becomes more essential because it is how you whisper rather than shout. Furthermore, legal requirements — laws and regulations — make it so that there are risks associated with losing data, which means there are real legal risks to anyone getting access to data you have.

Sadly, there are many things that are important, things we all should be doing that we're not as good at as we should be: flossing our teeth, changing the oil in the car every 3000 miles, and making sure that customer spreadsheet is encrypted on the laptop. If you are facing the difficulty of convincing yourself or others why you should be using cryptography, here are some stories you can use as ammunition.

Into the Breach: Horror Stories

Let's look at some recent security breaches cryptography could have prevented, at best, or turned into minor nuisances, at worst. I picked stories that go beyond the recent stream of data breaks and backup losses. These are all stories that very likely you didn't hear about because they happened *before* data breaches became national news.

Stolen Laptops

Stolen laptops are one of the most common sources of potentially catastrophic security lapses. Laptop thefts constitute about 48 percent of all computer thefts, followed by desktops at 26.7 percent and handheld computing devices at 13.3 percent[1].

Total losses from laptop thefts amounted to more than $6.7 million in 2004, and that figure covers just the cost of the hardware, not the value of the data the computers contained[2]. In fact, PGP software creator Phil Zimmermann has had two of his laptops stolen in train stations. Happily, Zimmermann practices what he preaches, so the thefts of his laptops ended up being annoyances, not catastrophes. Potential targets will only continue to increase: IDC reported that in 2005, over 50 percent of the PCs sold in the United States were laptops, up from the 29 percent reported in 2004.[3] This is an important threshold; over half of the computers people use are easily carried.

No one wants to lose a laptop; more and more of us have significant parts of our business and lives on our laptop. But if the data on the laptop has been encrypted, a theft is closer to an annoyance than a catastrophe. Consider these incidents where laptops containing unprotected data were stolen:

- In June 2004, University of California, Los Angeles, representatives warned 145,000 blood donors they could be at risk for identity theft due to a stolen university laptop. Thieves broke into a locked van in November 2003 and grabbed a laptop with a database that included names, birth dates, and Social Security numbers for all blood donors. The database did not include medical information other than blood type, and university officials did not recognize the significance of the loss and the potential for identity theft until the matter came up in a security audit in May 2004.[4]

- In May 2004, a laptop containing information on as many as 100 ongoing US Drug Enforcement Agency (DEA) investigations was stolen from the trunk of a car of an auditor for the US Department of Justice's (DOJ's) Office of the Inspector General (IG)[5]. The auditor then changed his story, claiming he had thrown the computer into a dumpster after he had accidentally damaged it. Regardless, the laptop contained 400 pages of case-file data that would make it possible to guess the identity of informants. Note that this incident occurred two years *after* the DOJ IG issued a report slamming the DEA and the FBI for shoddy data-security practices.

- In March 2004, a backup hard drive (not a laptop, but might as well have been) in the process of being driven to a bank for safekeeping was stolen out of a Spotcheck employee's car[6]. Spotcheck is a contractor that performs health insurance eligibility status for major US

[1] "Laptop locks: easy to use, easy to pick," *St. Paul Pioneer Press*, September 8, 2004.

[2] Computer Security Institute, *2004 CSI/FBI Computer Crime and Security Survey*

[3] Fitzgerald, Michael, "How to Stop a Laptop Thief," *CIO Magazine, December 8, 2004*: <http://cio.idg.com.au/index.php/id;1973406143;fp;4;fpid;18>

[4] Becker, David, "UCLA laptop theft exposes ID info," *CNet News.com*, June 10, 2004: <http://news.com.com/UCLA+laptop+theft+exposes+ID+info/2100-1029_3-5230662.html>

[5] "Missing: A Laptop of DEA Informants," *Newsweek*, June 7, 2004: <http://www.msnbc.msn.com/id/5092991/site/newsweek/>

[6] Lazarus, David, "Window smashed, data lost," *San Francisco Chronicle*, May 12, 2004: <http://www.sfgate.com/cgi-bin/article.cgi?file=/chronicle/archive/2004/05/12/BUG8O6JPV71.DTL&type=business>

health insurance companies nationwide such as Blue Shield and Cigna. In this case, personal and medical information on the 100,000 members of the Alameda Alliance for Health was stolen along with that hard drive.

- In February 2004, a laptop containing the names, addresses, and Social Security numbers of thousands of Wells Fargo mortgage customers nationwide was stolen from the unlocked car of employees stopping for gas in the Midwest[7].
- In January 2004, two laptops containing the names, addresses, dates of birth, Social Security numbers, credit scores, marital status, and genders of 200,000 customers of GMAC Financial Services was stolen from an employee's car in Atlanta[8].
- In December 2003, a laptop containing the names, addresses, and Social Security numbers of about 43,000 customers was stolen from Bank Rhode Island's principal data-processing provider. In response, the bank's CEO said the IT department planned to install encryption and fraud-detection software on all its computers[9].
- In November 2003, a laptop containing names, addresses, and Social Security numbers of Wells Fargo home-loan clients was stolen from the office of a consultant hired by the bank[10]. The thief, when caught, had a history of manufacturing fake IDs.
- In September 2000, the CEO of wireless giant Qualcomm, Irwin Jacobs, had his laptop stolen while addressing a meeting of the Society of American Business Editors and Writers[11]. Jacobs said that the machine contained "everything" from corporate information to several years' worth of email; he was 30 feet away from the machine when it disappeared.

Insecurely Protected Network Resources

Even enterprises that should know better (such as one of the top computer-science schools in the world or the world's largest software manufacturer) or that proclaim their customer data is secure (such as online stores) often are remarkably careless. Encryption of stored data could have prevented security breaches such as the following from happening:

- In August 2004, a hacker broke into a University of California, Berkeley, computer containing a database with the names, addresses, Social Security numbers, and dates of birth of 1.4 million caregivers and care recipients who had participated in California's In-Home Supportive Services (IHSS) program since 2001.[12]

[7] Lazarus, David, "Car thief whisks off Wells data," *San Francisco Chronicle*, April 16, 2004: <http://sfgate.com/cgi-bin/article.cgi?f=/c/a/2004/04/16/BUGH8650141.DTL>

[8] McDougall, Paul, "Laptop Theft Puts GMAC Customers' Data At Risk," *InformationWeek*, March 25, 2004: <http://informationweek.securitypipeline.com/news/18402599;jsessionid=YWJ4ORVP2WZQIQSNDBGCKHY>

[9] Mearian, Lucas, "BankRI customer information stolen along with laptop," *Computerworld*, December 19, 2003: <http://www.computerworld.com/securitytopics/security/story/0,10801,88443,00.html>

[10] Lazarus, David, "What's Next for Wells," *San Francisco Chronicle*, December 21, 2003: <http://sfgate.com/cgi-bin/article.cgi?file=/chronicle/archive/2003/12/21/BUGE73RAKL1.DTL>

[11] "Qualcomm CEO Loses Laptop," *Wired News*, September 18, 2000: <http://www.wired.com/news/business/0,1367,38855,00.html>

[12] Claburn, Thomas, "Break-In At Berkeley May Have Compromised Data Of 1.4 Million Californians," *InformationWeek*, October 20, 2004: <http://informationweek.securitypipeline.com/news/50900323>

- In February 2004, chunks of Microsoft Windows 2000 and Windows NT source code were posted on the Internet. Microsoft had not authorized this publication of more than 600 megabytes of its operating system — a field day for people interested in examining the software for security-related bugs[13].
- In August 2003, a hacker broke into the server at the Bancroft Library at the University of California, Berkeley, gaining access to the names, addresses, and driver's license numbers of 17,000 library users from all over the world. The Bancroft is a repository for rare books and historical artifacts, not a place whose use would usually lead people to feel that they might be at risk for identity theft. Data going back 12 years was stored on the library's server[14].
- In February 2002, purely on a lark, a young computer programmer named Jeremiah Jacks was able to pull down every name, credit card number, and associated expiration date for all 200,000 customers on the Guess.com web site[15]. Because Guess.com had claimed that all its customer data was securely encrypted at all times, the company was fined by the US Federal Trade Commission (FTC), required to create and maintain an independently audited security program for 20 years, and prohibited from making any claims about the security of its data.

  Although it was not fined by the FTC, in June 2003, Jacks found that PetCo.com had the same database vulnerability as Guess.com[16].

Javelin Strategy and Research and the Better Business Bureau released their *2006 Identity Fraud Survey Report* in January 2006. This report updates a report made in 2005 by Javelin and the BBB as well as an FTC survey report from 2003 [IDTheft].

The survey report includes information such as:

- The number of adult victims in the US decreased from 10.1 million in the 2003 report to 9.3 million in the 2006 report.
- However, the total one-year fraud amount increased from $53.2 billion to $56.6 billion from the 2003 report to the 2006 report.
- The mean amount stolen from victims rose from $5,249 to $6,383.
- The time it took each victim to resolve the fraud also rose from 33 hours in the 2003 report to 40 hours in the 2006 report.

A Few Words About Identity Theft

The definition of identity theft has changed in the last few years, particularly as a result of the US government passing laws to fight it. Under the legal definition, identity theft is nearly any misuse

[13] Lemos, Robert, "Microsoft Probes Windows Code Leak," CNet news.com, February 12, 2004: <http://news.com.com/2100-7349_3-5158496.html>

[14] Lazarus, David, "Online breach at Bancroft," *San Francisco Chronicle*, November 23, 2003: <http://www.sfgate.com/cgi-bin/article.cgi?file=/chronicle/archive/2003/11/23/BUG5D37C7T1.DTL>

[15] Poulsen, Kevin, "Guesswork Plagues Web Hole Reporting," *SecurityFocus*, March 2, 2002: <http://www.securityfocus.com/news/346>

[16] Poulsen, Kevin, "PetCo Plugs Credit Card Leak," *SecurityFocus*, June 30, 2003: <http://www.securityfocus.com/news/6194>

of an identifier you might have. So if someone breaks into your iTunes account and buys a lot of music charged to your credit card, that's legal identity theft.

However, there is also a larger crime that we call identity theft. In this form of identity theft, the criminal doesn't merely use your credit card number, but also gets entire credit cards in your name, under your credit records, and frequently they are sent to some other address. Let's call them little-i and big-i identity theft. In big-i identity theft, it's a surprise to you that someone has been pretending to be you, often for months. You find out about it because angry creditors trace the false you to the real one.

Big-i identity theft is still primarily a low-tech crime. The criminal often has at least met you or seen you and gets information about you from your trash, bills, pre-paid credit card offers, and similar information. Encryption can't stop a dumpster-diver, but a shredder can. You need a shredder, too. Not only will it protect you, but shredding junk mail is one of life's great small pleasures.

Laws and Regulations

Nearly any discussion of cryptography includes a section on laws and regulations that hover around its use. However, that discussion has changed radically. In years past, the discussion would have been about legal and regulatory *limitations* on the use of cryptography. Although these have not completely gone away, they are no longer an impediment to using encryption. Those remaining restrictions will be discussed below. One of the most significant changes in the use of cryptography is that in the last few years, laws and regulations have arisen that promote its use, cajole organizations to use it, and in some cases *require* its use.

Following is an overview of a number of regulations. Please note that as time goes on, there are far more likely to be more of them rather than fewer.

Privacy Regulations

- The *European Union Privacy Directive*, also known as the *Data Protection Directive* (DPD), mandates that all EU member countries enact comprehensive legislation protecting personal data. It also requires that non-EU member countries doing business with member countries follow minimum standards for safeguarding personal data. The seventh of the DPD's eight key principles requires that personal data must be secure. It is the world's model for all privacy laws. [EUDPD]

- Canada also has some of the best data privacy laws. There are have two key laws, the *Privacy Act* and the *Personal Information Protection and Electronic Documents Act*. Individuals are also protected by the *Personal Information Protection and Electronic Documents Act* or PIPEDA. [CANPRIV]

- The *National Privacy Act* of Australia and *National Privacy Principles* protect personal information possessed by government agencies and safeguards the use and collection of tax-file numbers. [AUPRIV]

- The US *Health Insurance Portability and Accountability Act* (HIPAA) requires that the Department of Health and Human Services (HHS) establish security standards and privacy guidelines for the electronic exchange of health insurance data and personally identifiable information.

  Note that HIPAA is not strictly a privacy law; the "P" in it stands for *Portability*, not *Privacy*. Its goal is to encourage standard, portable transfer of health and insurance records. To make portability a step forward rather than a step backwards, there has to be privacy and security. The HIPAA Final Security Rule mandates security policies and procedures and explicitly suggests the use of encryption, "for transmitting electronic protected health information, particularly over the Internet." HHS encourages the use of encryption as part of HIPAA's Technical Safeguards.

- Japan's *Personal Information Privacy Act* (PIPA) applies to any company that has offices in Japan and has the personal data of at least 5,000 people. Personal data includes the person's name, address, sex, date of birth, telephone numbers, and email address. PIPA requires that a company have a Chief Privacy Officer who manages compliance and sets fines of ¥300,000 or jail sentences of up to six months for violations. [JPPRIV]

Compliance Regulations

- *Basel II (The New Capital Accord).* Created by the Bank for International Settlements, this set of regulations was designed to reduce risk in the operations of international financial services providers (FSPs) in Europe, the Americas, and Asia. Among other things, Basel II sets standards for measuring and improving FSP information technology security.

- *Sarbanes-Oxley Act (SOX).* SOX is a US corporate-accounting legislative cleanup effort, demanding greater transparency and accountability in publicly held U.S firms and their auditors. SOX Section 404, Management Assessment of Internal Controls, stipulates that internal IT systems be tested (and upgraded, as needed) for soundness and security. To meet the SOX Section 404 challenge, the Business Security Alliance formed the Information Security Governance Task Force, which identified IS0 17799, a security-controls standard that met the requirements of the Federal Information Security Management Act (FISMA).

  Sarbanes-Oxley does not specify how to adequately secure financial reporting information, nor does it specifically mandate encryption solutions. However, both ISO 17799 and the security controls for FISMA include recommendations for encryption and digital signature controls to help prevent the loss, modification, or misuse of system data.

- *Gramm-Leach-Bliley* (GLB, also known as the *Financial Services Modernization Act*) is another US compliance law. It requires that US financial institutions maintain the confidentiality and security of customer information, with particular emphasis on protecting data from hackers.

  Although GLB guidelines do not require encryption of customer information, if the financial institution concludes that encryption is appropriate, then it must implement it. The Federal Financial Institutions Examination Council (FFIEC) recommends encryption as an appropriate risk-mitigation technology. Organizations that do not adopt encryption to the degree expected by the FFIEC have the burden of demonstrating that they considered it and showing why they decided not to use it.

- US Federal Drug Administration (FDA) *Title 21 Code of Federal Regulations Electronic Records; Electronic Signatures (CFR Part 11)* is part of the *Government Paperwork Elimination Act*, 21 CRF Part 11 promotes the use of electronic signatures in FDA-regulated industries and specifies how electronic records must be handled by medical device, drug, and biological manufacturers. Part 11 focuses on authenticity and confidentiality of data and requires secure and validated data management.

Breach Notification Regulations

The model law that started breach notifications is California Senate Bill (CA SB) 1386, the *Database Security Breach Notification Act*, commonly called SB 1386. It amends the California Civil Code so that any organization doing business within California, whether public or private, and whether or not located in California, is required to notify any affected California residents of any relevant security breaches within their organizations. CA SB 1386 applies to unauthorized dissemination of driver's license, Social Security, credit card, bank account, and library card numbers or similar data.

Although SB 1386 does not specifically require organizations to encrypt personal information, it states that the law does not apply to data that has been encrypted. In other words, if personal information is encrypted, organizations can avoid having to notify customers of a security breach.

SB 1386 is the reason that there were so many reports in 2005 of companies losing data, often through no overt act of their own. Sometimes it has been from a laptop or computer stolen just for the money[17]. However, SB 1386 has been a very significant law. It requires only notifying the people whose data was lost (and permits them to sue), without any penalties other than public embarrassment. Additionally, it provides a "get out of jail free card" if the data that was lost was encrypted. This requirement has gently pushed businesses to take breaches more seriously.

As I write this, twenty-three states in the US have enacted similar laws. There is discussion in Congress for a national version. There are now similar laws in Australia, Japan, and other countries. The effect of SB 1386 is such that I cannot accurately describe the world situation except to say that the worldwide trend to notification is such that it's a good idea to encrypt sensitive data.

Laws and Regulations Limiting Cryptography

Prior to the late 1990s, cryptography was considered a military technology. It was regulated the same way that arms were regulated, under the very same regulations. This practice gradually changed over a few years, particularly in the United States and France, which had the tightest restrictions.

Cryptography is still considered a "dual-use" technology, one that can be used for both civilian and military purposes. It is no longer considered a munition, but merely something that can be used for

[17] My own health association in California "lost" some 750,000 personal records as part of computer thefts. It turned out after an investigation that it was an inside job and the computers were stolen as things — it wasn't the data at all the thieves were after, but the money from selling the "used equipment." But when it received national attention, the plot came to light and the thieves even turned themselves in.

both good and evil, just like nuclear fuel and video games[18]. However, there was a huge change in regulations governing the use of cryptography in 1999 and 2000. France, which had previously been the most restrictive country in regards to the use of cryptography, became one of the most liberal. The United States had no restrictions on the *use* of cryptography, but had them on the *export* of cryptography. The US export regulations were dramatically liberalized in 2000.

Worldwide, export of cryptography is still controlled under the "Wassenaar Agreement" an international set of agreements that control dual-use technologies. There are continual changes to the agreements, and in general, the laws and regulations continue to move towards sanity, despite international terrorism.

At this writing (late 2006), cryptographic software such as PGP software can be freely sold and downloaded everywhere but the US's list of seven restricted countries: Cuba, Iran, Iraq, Libya[19], North Korea, Sudan, and Syria. The remaining barriers to the free use of cryptography are less export restrictions from the US, but import restrictions into countries that value wholesale spying on their citizens (such as China).

The laws and regulations limiting cryptography still exist, but for practical purposes are a thing of the past.

[18]Very fast computers are considered dual-use, and things being what they are, the fastest computers available end up being used for game computers. This practice leads to the ironic situation that game consoles end up with some of the most bizarre export issues.

[19]Libya is being removed from many restrictions as a result of normalized diplomatic relations. By the time you read this, it may be removed from all restrictions.

Chapter 3

An Inadequate History of Cryptography

> *History is written by the historians.*
> – Sir Leigh Teabing (attributed)

The definitive *adequate* history of cryptography is David Kahn's *The Codebreakers: The Story of Secret Writing* [KAHN]. *The Codebreakers* inspired a whole generation of cryptography and security experts. It covers the history of codes and ciphers, making and breaking them from the Egyptians through WWII. Kahn's other books are also well worth reading for more detailed looks at cryptologic history.

However, Kahn trails off in *The Codebreakers* precisely where cryptography starts to get interesting in today's practical aspects of it. Even other adequate histories such as Singh's *The Code Book* [SINGH] do not tell someone with practical interests and needs where we are and how we got here. That is why I offer this *inadequate* history of cryptography. I will gloss over many things that you can read elsewhere in luscious detail. I'm not going to summarize Kahn. I *am* going to talk about things he couldn't, and be biased, glib, and on occasions puckish.

Human Cryptography

Cryptography is nearly as old as writing. No one knows exactly when it started. My opinion is that about the time three people knew how to read and write, two of them wanted to write something that the third one couldn't read. I can only hypothesize, but my bet is that it was the second and third scribes who realized that if they got clever, they could take old smarty-pants down a peg or two. Somewhat after this, kings and rich merchants realized the power of written messages[1]. If you wrote something down and sent it to your ambassador, general, buyers, and other trusted people, the messenger (who was illiterate, because there were only a few hundred people in the world who could write) couldn't read it. If the messenger was intercepted, the message couldn't be beaten out of him, because the messenger didn't know what the message was.

[1] Governments and merchants have always funded the development of technology. They transformed writing itself from merely being a generally accepted accounting trick to the first form of telecommunications. The value of communications at a distance has not been lost on government, military, or business ever since.

Of course, once it became relatively common for there to be experts who could decode chicken scratch on tablets, the technologists had to come up with secret writing that not everyone can read, and thus cryptography became a discipline and technology, rather than a parlor trick among scribes.

Just as biologically, ontogeny recapitulates phylogeny, so it happens with cryptography. When I was a kid, I learned that if you passed notes in class with secret writing, they couldn't be read aloud to everyone when intercepted by an adversary. I learned to write notes in Greek letters – which is a form of coding rather than ciphering, but it's a good start at secret writing. When you start at it, it looks easy enough, but you run into all the classic coding problems. Immediately, I had to cope with the fact that there is no Greek character for "J," two for "O," and two ways to code a "C." Other people would immediately note other coding issues, such as one glyph for "TH" and "CH" but no good way at all to code an "SH."

Bruce Schneier somewhat famously said that there are two types of cryptography: the sort that will stop your little sister from reading your files and the sort that will stop national governments. Robert Morris Sr gives the advice that there are two factors in information security – the desire of the adversary to read your files, and how much you care that they do. If your adversary doesn't particularly want to read your files, you don't need to spend much effort protecting them. If your adversary very much wants to read your files, then you must spend a lot of effort to protect those files. Taken together, these observations suggest that there's actually a third sort of cryptography that Bruce didn't mention — the sort of cryptography that is inadequate to stop your little sister (who desperately wants to read your files), but is completely adequate to stop your teacher (who only wants you to stop passing notes in class)[2]. Writing in Greek letters is this third form of cryptography.

With the help of another annoying teacher and my sympathetic mother, I moved on to another form of coding — Gregg shorthand. Shorthand has a number of advantages as a coding mechanism. First, the glyphs it uses are not letters, but a delightful set of squiggles. Second, it isn't an alphabet; it is a *phonetic* coding system. The adversary in this case was someone who wanted to teach us all to take notes, but what he called notes most of us would call dictation. He graded on the accuracy and completeness of the dictation we students took. Writing in shorthand allowed me to take complete notes that he couldn't read, but I could prove were complete[3].

Coding as a general technique is doomed to fall to the dedicated little sister. In its most advanced forms, it worked reasonably well for thousands of years against national governments. To break the code, the attacker merely has to deduce or obtain a copy of the coding system. Unfortunately, that's all they have to do. Little sisters will always have an easier time of this than national governments. Techniques that look clever, such as inventing your own symbols, turn out to be mediocre in practice.

And so just as people did in millennia past, brothers will progress up the phylogeny from codes to ciphers. In history, Julius Caesar famously and effectively used ciphers. The one that he used is very simple — shift the letter being used by three. Thus, "A" becomes "D," "B" becomes "E" and so on. The key here is 3, because it's simple to make it be a different number. Now, this isn't a very good cipher because the *key space*, the total size of possible keys, is only 26[4] and it is thus relatively easy for a cryptanalyst to write out all the possibilities.

[2]It also suggests that Bruce doesn't have a little sister, or at least not one like *my* little sister.

[3]Note, however, that this method is *still* vulnerable to the little sister adversary who has access to both mother and shorthand textbooks

[4]Smaller for Caesar, as the Romans had fewer letters, notably missing "J" and "U."

The foregoing describes the way that ancient cryptography progressed — and often did not progress. Cryptography was something that a person did. An expert cryptographer would devise ways to code and encipher. Expert cryptanalysts would devise ways to translate the codes and break the ciphers. Some cryptographers would write books about principles of code making. Some cryptanalysts would write books about the principles of code breaking. Often, these books themselves would be secret or in limited distribution. Often, talented cryptanalysts would be accused of being in league with the devil, having magical powers, or causing hunts for spies inside the group that was coding.

Human cryptography is also limited by the complexity of the system. In all these systems, there is the danger of miscoding, misciphering, or both. It reached its pinnacle around the time of WWI [AEGEAN]. By this time there were books that described how systems worked and how they could be broken. The best national cryptologic departments had codified their techniques well enough that they could teach them as courses to new recruits. The knowledge itself was considered secret and proprietary, as there were many nations that did *not* have the same levels of expertise as the best ones. Roughly by the end of WWI, cryptography had ceased to be a black art, and was becoming a science. The science progressed further through the end of WWII, during which human cryptography was actively used. The techniques of human cryptography are likely even used by some computer programs, which is why the cryptography in these systems is trivial to break.

A cryptanalyst typically breaks a message by relying on the fact that some letters appear more often than others. By looking at the frequency of the cipher characters, the cryptanalyst guesses that the distribution in the ciphertext will approximate the distribution of letters in the unencrypted plaintext. The order of most commonly used letters in English is ETAOIN SHRDLU. In fact, statistical variations in encrypted data is how almost all cryptanalysis is done.

The cryptographer can fight this statistical cryptanalysis through a number of techniques:

1. **Make the messages smaller.** If the messages are small, then the cryptanalyst has less statistics to gather. This is easier said than done. After all, the whole point of encryption is to be able to communicate secretly, and not communicating ruins the point. The logical conclusion of making messages smaller is not to send them at all.

2. **Smooth out the statistics.** The cryptographer can change the statistical count of letters by having extra copies of common characters as well as noise characters that don't mean anything. For example, instead of numbering letters from 1 to 26, why not take digits from 1 to 100 and have a few "E"s but only one "Q." This helps, but really only forces the cryptanalyst to collect more text before applying statistics. Worse, human beings are bad at using the multiple characters well, and they don't switch them around enough.

3. **Use symbols not just for letters, but for whole words.** This technique is why codebooks were so popular in advanced human cryptography. This approach makes the cryptanalyst's job harder, because they have less to go on. Even better, combine this technique with the one above by having multiple codes for common words.

4. **Encipher only parts of the message.** This cuts down the amount of ciphertext the analyst gets, but on the other hand yields context. Often what the cryptanalyst gains from the context renders the cryptography moot. A message like, **Diplomatic bigwig from 9001 met with trade minister from 9049 and discussed the ongoing tensions in 9964**, combined with

a newspaper might give away that 9001 means the United States, 9049 is Germany, and 9964 is Iraq. These snippets will aid the cryptanalyst later, especially since all those start with 9000.

5. **Use multiple ciphers.** If different parts of a message are encrypted differently, it makes the analyst's job harder. Again, this approach makes life harder for both communicating parties.

One of the more interesting forms of advanced human cryptography is the Jefferson Wheel [JWheel], named for its inventor, the US philosopher and President, Thomas Jefferson. It consists of a set of rings, each with a scrambled alphabet. The key is the construction of the wheel. I encipher by dialing my message on the rings and sending you the row of letters above my message. It thus uses a whole set of ciphers, one on each ring, and gets added security this way.

All these techniques, however, make human cryptology a contest between the cleverness of the cryptographer and the cleverness of the cryptanalyst. Often, as well, being too clever in cryptography can actually aid the cryptanalysis. [URBAN]

Machine Cryptography

After WWI, a number of people started thinking of new ways to do cryptography. Cryptography was becoming a science, and skilled experts had broken even the most sophisticated human cryptographic systems [ADFGVX][5]. The goal, of course, was to create a cipher that was unbreakable. The means to that goal seemed to be to create cipher machines that would take the same basic principles that human cryptographic systems used, but with complexity and reliability that would make breaking such a cipher beyond the ken of any human cryptanalyst.

The most famous of these machine cryptosystems is the Enigma machine. It was built by Arthur Scherbius and Richard Ritter in 1918. Scherbius's original business plans were to sell the Enigma for encrypting messages for banks, lawyers, and the like. It was not until years later that the German government saw the Enigma, liked it, and began its use to encrypt messages. Although it was the machine used most often by German agencies during WWII, it was not the only — or even the most complex – device used. Nonetheless, a variety of the Enigma was regularly used by major governments until the early 1990s. [EnigmaBP] [EnigmaDM]

Machine cryptosystems generally followed a basic design based on a series of gears, wheels and rotors that stepped as they encrypted texts. The Enigma used three or four of these rotors originally, growing to eleven in the NEMA (for Neue Machine, or New Machine). This class of cipher machines is often called *rotor machines*.

The rise of machine cryptography was a success. You may be surprised to read that; the Enigma is famously broken by the Allies as was the Japanese PURPLE machine [MAGIC]. The American Hagelin M-209 machine was also broken [Hagelin]. Machine cryptography succeeded because it put the human cryptanalyst out of business. Now, there were many human cryptanalysts still employed,

[5] ADFGVX was also different from what I have described previously in that it not only substituted letters for other letters, but also rearranged the letters in permutations, using two ciphertext symbols to denote a plaintext symbol. It is called ADFGVX because it used only those six letters in pairs.

and of course humans broke machine cryptography even as they improved it. But cryptanalysts changed as a result of machine cryptography. Before the machines, a cryptanalyst typically came from a language department of a university, not a math department. They would be people skilled in languages, word games, puzzles, anagrams, and so on. The rise of the machines meant that cryptanalysts started to become statisticians, mathematicians, mechanics, and engineers.

There is also one other thing that machine cryptography did: it forced the invention of the digital electronic computer.

Like most things, electronic computers weren't created all at once. A number of computers were built mechanically or electromechanically with relays and telephone switching equipment. These devices were used by the cryptanalysts faced with machine cryptography because they had to. The problems machine cryptography created were just too large to be solved by hand. Electromechanical systems that weren't quite computers broke the Enigma[6] as well as the Japanese PURPLE and the Hagelin machines. The Lorenz machine, a cipher machine used by the Germans for high-level communications, needed more oomph than the mechanical systems had. Therefore, the first programmable, fully electrical, digital computer was created to cryptanalyze the Lorenz machine. That computer was called Colossus.

Colossus is not fully programmable the way modern computers or Konrad Zuse's machines [ZUSE] are. You program Colossus by flipping switches to program it, and all of the programmable things are related to breaking codes in general and the Lorenz machine in particular. But what it may have lacked in general-purpose utility it made up for in speed. Colossus was fast. A Colossus emulator running on a modern Pentium needs about a 2GHz machine to keep up with a real Colossus. In contrast, Zuse's machine, which was programmable enough to play chess[7], was capable of about 2000 calculation per hour, not two billion per second.

There were a total of ten Colossi made. After the end of the war, two were sent to Cheltenham from Bletchley Park, where they were in use until the early '60s. The others were destroyed under Churchill's orders and the plans burned. Colossus was something of a myth for fifty years, talked about vaguely in both computer science and cryptology circles, especially because it was built directly from Alan Turing's work. It was finally rebuilt by a team led by Tony Sale of the Bletchley Park Trust [BPTrust] and Codes and Ciphers Heritage Trust [CCTrust] over the course of a decade.

The effect successful machine and computer cryptanalysis had on the world was significant: all forms of cryptology were considered pretty much the domain of national governments for the next few decades, and national governments were resentful of any civilian dabbling in the discipline.

To a certain extent, this is understandable, given the effect machine cryptology had on WWII. In fact, machine cryptography was quite successful even after the war. The Swiss, for example, made an eleven-rotor version of the Enigma called NEMA, and it was actively used until the early '90s. Rotor-based ciphers still affect the design of stream ciphers to this day.

[6] An excellent movie to watch is the movie *Enigma* [EMovie]. It is a mystery set in Bletchley Park, where the British cryptanalysis took place. The plot itself isn't historically accurate, although it is still an engaging mystery tale. But unlike most movies set in some historical or technical setting, *Enigma* is accurate in historic details — cryptographically, anyway. It was made with the direct help of the Bletchley Park Trust. One of its producers, Mick Jagger, is a cryptography buff and has a collection of cipher machines. Consequently, if you want to see how cryptography, cryptanalysis, and the rest of signals intelligence were done in that era, watch it and enjoy.

[7] Zuse boasted of this, and although the machines he was building at the same time as Colossus did not have the capability of it, he did write a chess-playing program about sixty pages long.

Eventually, the widespread use of computers brought another change in cryptography, the one with which we're concerned. As interesting as they are, human and machine cryptography are primarily historic, not of present *direct* practical use.

Computer Cryptography

I somewhat arbitrarily picked the end of WWI to mark the beginning of machine cryptography. I'll pick the somewhat arbitrary date of 1975 to mark the beginning of the present age of cryptography, the age of computer cryptography.

I pick 1975 for two reasons: the development of public-key cryptography, and the development of DES, the Data Encryption Standard. Each of these, in its own way represent basic changes to cryptography itself, the way it is used and thought about, and even the language we use to describe it.

Public-Key Cryptography

Throughout the history of cryptography, there has been one problem that has made the practical use of cryptography difficult and unwieldy — the problem of key distribution. The best cipher is only as strong as its keys. If the adversary can deduce or obtain the keys being used, then they can read the ciphertext regardless of the strength of the cipher[8].

Key distribution limited the use of cryptography precisely because it was so hard. If you've ever seen an old spy movie with someone carrying a briefcase handcuffed to his wrist, you've seen the key distribution problem in popular culture.

In the early 1970s, a number of young mathematicians and cryptographers started to think about the key distribution problem and how to solve it. Ralph Merkle, Whitfield Diffie, and Martin Hellman each did individual work on it and also worked together on various schemes. Interestingly, Ralph Merkle started out by trying to prove that it was impossible to finesse the key distribution problem. The three of them came up with various schemes to do this; one of them, Diffie-Hellman, is actively in use today. A couple of years later, three other mathematicians, Ron Rivest, Adi Shamir, and Len Adleman, came up with the RSA scheme named for them[9].

Public-key cryptography changes the way cryptography is done and solves the key-distribution problem by making a scheme that uses two keys, rather than one. Before, all cryptography had a single key and that key was used to encrypt the data as well as decrypt it. Public-key cryptography uses a pair of keys: one that encrypts the data and one that decrypts the data. Furthermore, you cannot deduce the decryption key from knowing the encryption key. Because knowing the

[8] In fact, cryptanalysis is often at its best figuring out the keys being used, as opposed to actually breaking the cipher itself.

[9] It is claimed that a few years earlier, a few cryptographers at the British GCHQ came up with a public-key cryptography scheme similar to RSA, but they neither published it nor put it into practice, and so they remain an uncredited footnote. CESG had at one time some information about "non-secret encryption" as they called it, but I could not find them as I wrote this, and this makes their non-verifiable claim even less verifiable. The best quick description I have available for you is Bruce Schneier's at [NSENC].

encryption key doesn't let you know the decryption key, there is no reason for it to be a secret. You might as well publish it in a newspaper, skywrite it, or scrawl on some bathroom wall, "For a secret time, encrypt to this key."

This is why this new form of cryptography is called *public-key* cryptography. One of the keys can be completely public without hurting the security of the system. With public keys, we put the suave guy with a briefcase handcuffed to his wrist out of business. We don't need him to hand out the proper keys to each of us — I just use your public key and you use mine.

As I've said before, much of cryptography is intuitive, because we humans have been doing cryptography of one form or other almost as long as we've been writing. It's perfectly reasonable to look at public-key cryptography like it's something weird, because it is.

First of all, public-key cryptography creates a whole host of language problems. The encrypt key of this system is pretty straightforwardly called the *public* key. The decrypt key, however, is sometimes called the *private* key and sometimes the *secret* key. In my opinion, *private* is a better word than *secret*, but that means that if we want to abbreviate them when we talk about the recipes to use them, we're talking about two things that start with P. Therefore we have to call them Pu (public) and Pr (private), which is inconvenient, especially when scribbling on napkins in restaurants, as all good systems designers do. Instead, we'll call them the public key (P) and the secret key (S), so we can use different letters.

There is one more language problem. What do we call the plain old, ordinary cryptography? We've already used the words *public*, *private*, and *secret*. Another way the two types of cryptography have been described is to call the original cryptography *symmetric-key* cryptography (because the single key both encrypts and decrypts) and *asymmetric* cryptography for public-key cryptography because, well, it's not symmetric.

These may not be the best terms, but if you read more, you'll see all of them used. From here on, I'll differentiate between the broad categories of *public-key* and *symmetric* cryptography, and within public-key cryptography, I'll refer to *public* and *secret* keys. I find these terms to be the least tongue-tying, even if they mix metaphors.

How does public-key cryptography work? It is, as I've said, counterintuitive and downright screwy to think that you can have these two keys that are joined yet unrelated. This is why it it is such a recent development — and also because public-key cryptography is so hard to do that it wouldn't be practical at all if it weren't for computers.

Public-key cryptography is based on the idea that there are some mathematical operations that are relatively easy to do, but hard to undo. Anyone who has ever had to do math by hand understands this intuitively. Multiplying is easy, long division is hard. Factoring numbers, well that's even harder. Taking numbers to powers is easy, but taking roots and logarithms is hard. The different public-key algorithms are based on this core idea that there is this asymmetry in doing and undoing. By hand, it's harder to factor 391 than it is to multiply 17 and 23.

Unfortunately, even (especially?) on computers, it's not *that* much harder to do these hard operations than the easy ones. So we have to use very big numbers. These days, the smallest keys we recommend using with the PGP software are 1024-bit keys, which are around 300 digits long. We strongly recommend that you use keys that are are 2048 to 4096 bits long, which means they are about 600 to 1500 digits in length!

Working with numbers this big would not be possible without computers, and this is why public-key cryptography is computer cryptography.

The Rise of Standard Cryptography

As I mentioned earlier, the other thing that happened in roughly 1975 was the development of the Data Encryption Standard, or DES. Just as business needed to communicate secretly in machine cryptography, this same need became apparent with computer cryptography. In 1973, the US government requested help in devising a cipher to be used throughout the federal government for unclassified data, and they issued a second request in 1974. IBM responded to this request with a cipher based on a previous cipher called Lucifer. As part of the process of creating DES, the National Security Agency (NSA) changed DES, shortening its key from an eight-byte (64-bit) number to a seven-byte (56-bit) number, and also changing some of its internal structure (specifically, data structures called S-boxes). Furthermore, the NSA did not tell anyone why it made the changes it did, merely saying that it had improved DES. DES was approved as a US Government Federal Information Processing Standard (FIPS), FIPS-46, in 1977.

The NSA's changes to DES made it controversial. How could shrinking the key from 64 to 56 bits "improve" it, unless, of course you think a smaller key is an improvement? Secrecy makes people suspicious, especially when that secrecy doesn't jibe with intuition. For many years, cryptographers had many not-very-polite opinions about DES. Despite a Senate Select Committee on Intelligence investigation into the matter, which concluded that there had been no flaw introduced into DES, civilian cryptographers presumed that not only had the NSA weakened DES, but it was quite likely that they had knowledge about DES's internal structure that would let it decrypt messages encrypted with DES with substantially less work than the rest of us could[10]. An internal secret flaw in a cryptosystem is called a *backdoor*.

This was not an unwarranted fear. People have created ciphers with backdoors in them, intentional flaws that allow people with knowledge of the flaw easily decrypt messages.

Nonetheless, it's important to note how important DES is.

Remember that for most of human history, cryptography was something that clever people did. Your nation's cryptographic prowess was measured by how smart your cryptographers and cryptanalysts were compared to the other side's cryptographers and cryptanalysts. During the machine cryptography age, this paradigm changed a bit: cryptographers made machines that did the cryptography, and if you had the right machine you had the right security. The idea that you could have *standard* cryptography is a radical shift in thinking. You can't have standard coding or steganography because if we all hide things the same way, we know where to look. The idea that you can build a cryptographic system that has public components (even if we don't know all the design principles) was a radical departure from conventional thinking. It derives from the thinking of the late-nineteenth century cryptographer Auguste Kerckhoffs and is today known as *Kerckhoffs's*

[10]There is a certain amount of debate as to exactly how much the NSA directly changed DES during its development. One member of the DES team, Walter Tuchman, has said the NSA didn't change a thing. Another member, Alan Konheim, claimed the S-boxes were completely changed. The Senate Committee that reviewed the controversy concluded that although the NSA had convinced IBM that the smaller key size was good enough, it only indirectly affected S-box design. I'm of the opinion that the NSA did more than indirect changes, but your opinion is as good as mine.

Principle — that in a good cryptographic system, the construction of the system itself should not be a secret, only the keys used in the system should be secrets [KERCKHOFFS].

DES did not quite meet Kerckhoffs's Principle, but it was a step toward it. National governments were loathe to give up their knowledge about cryptography. By 1991, there was open contempt for DES's quality because of the surrounding politics, yet civilian cryptographers were starting to make headway in cryptanalyzing it, and concluding that it might not be so bad, after all. In that year, cryptographers Eli Biham and Adi Shamir published a paper [DIFCRYPT1] that is a detailed cryptanalysis of DES. In it, they developed a new (for us) form of cryptanalysis called *differential cryptanalysis*. Using their new technique, they showed that the DES S-boxes were much more secure than they would be if they had been chosen at random; apparently, someone involved in the design of DES knew about differential cryptanalysis. They also showed that a 64-bit DES with random S-boxes is a weaker cipher than the 56-bit DES. Of course, it would have been even better to have longer keys and a stronger structure, but what's done is done. They opined that there appeared to be no intentional flaws in DES, despite what some people thought.

In 1994, one of the DES designers, Don Coppersmith, published the design criteria for the DES S-boxes and showed that indeed, IBM had known about differential cryptanalysis as they created DES. Coppersmith said that the NSA had persuaded IBM to keep that knowledge secret, because it is a powerful, general-purpose way to analyze a cipher. So it appears that the NSA and IBM each knew about differential cryptanalysis before Biham and Shamir discovered it, but that each independently discovered it.

DES is the most-studied cipher that currently exists. We know more about it than anything else, because the standards process focussed attention on it. With all its flaws, it's a pretty good cipher, especially for its era. Today, much of what we know about making ciphers comes from studies of DES. The biggest problem with DES is its key size; 56 bits is too small. When this became apparent, DES was used as a component of a new cipher, Triple-DES, composed of three operations of DES itself and either two or three 56-bit keys[11]. Of course, Triple-DES is three times slower than DES proper, and is somewhat tetchy to use; if for example, the same DES key is doubled or tripled, then Triple-DES is exactly Single-DES.

This situation led to another step toward true standard cryptography. DES was originally approved as a FIPS for five years. In 1983, it was extended for another five years. It was extended again in 1988 and again in 1998. By the time Biham and Shamir had published their cryptanalysis of DES, the five-year standard had been extended twice. Despite increased confidence in the DES design, its short keys still made it a target of derision and contempt. A replacement for it was needed.

The Advanced Encryption Standard

In the mid-'90s, NIST started thinking about what a replacement for DES needed to be. They spoke with cryptographers and engineers about requirements for a new standard cipher. Among the suggestions was that the next cipher should be selected by a competition, rather than by commission. In January 1997, NIST announced that they would start the transition away from DES toward an

[11]The same tripling can be done with just about any cipher to make it stronger. As it turns out however, you cannot double a cipher to make it stronger. There are a class of attacks called "meet-in-the-middle" attacks that make a doubled cipher no stronger than the base cipher itself.

Advanced Encryption Standard, and that there would be a competition for the AES [AESCOMP]. In September of that year, they announced the Request for Candidate Algorithms, giving the actual requirements for the AES. These included:

- Candidate ciphers had to be block ciphers, not stream ciphers.
- Candidates had to use a 128-bit block.
- Candidates had to use a minimum of 128-bit keys, but be expandable upwards. Specifically, candidates had to operate with 128, 192, and 256-bit keys.
- Candidates had to run at least as fast as Triple DES, preferably as fast as DES proper.
- The winning submission had to be free of intellectual property constraints. It was permissible to nominate a patented algorithm, but if it won, there had to be free licensing.
- DES was designed to last five years, and ended up lasting far more than that. At the time of the competition, everyone knew the earliest it would be retired would be 2003. Consequently, a replacement must take into consideration that it would be used for likely between twenty-five and fifty years.

Two of these main requirements are not a surprise. If a DES replacement is slower than Triple DES, people aren't going to use it. Likewise, if there are legal restrictions on its use, it isn't going to be used. But the two others were bold; ciphers of the time typically used 128-bit keys and a 64-bit block size[12]. Cryptographers were simply not working with the parameters required by AES.

A total fifteen ciphers were submitted as candidates for the AES. NIST held three conferences, the first being in the summer of 1998.

Looking back at the AES process, it was an amazing thing to do because it directly addressed the problem with DES. DES was a public-works project, but a secret public-works project. The requirements, processes, decisions, and debates around its design and construction were (and are) a secret. Even today, it is somewhat controversial to defend DES. An open, participatory, competitive process to select a new standard cipher was the best way to address those concerns.

The biggest problem, of course, is how to conduct such a competition. NIST said they would narrow the initial submissions to a set of five finalists, and then select the AES from those finalists. They said that while community input would be an important part of the selection, they would be making the selection rather than taking a simple vote. In my own conversations with the NIST people, they said that they took the comments of people who did not have a submission in the competition with more weight than the comments from the competitors, for all the obvious reasons.

[12]There are, of course, a number of exceptions. The Square cipher used a 128-bit block size. Also, there were a number of ciphers that had variable key sizes. But key sizes larger than 128 bits would not necessarily lead to greater security. Bruce Schneier's Blowfish cipher, for example, can use keys up to 448 bits, but Schneier himself does not recommend using more than 128. The RC4 cipher, which is still used in SSL connections, can use up to a 2048 bit key, but we know that it has at most about 600 bits of security no matter what key size you use — some cryptographers are much more cynical and opine that they would be surprised if it has as many as the usual 128. Nonetheless, it is important to realize that key sizes larger than 128 bits was more an intellectual flourish (or nose-thumbing towards governments as this *was* during The Crypto Wars) than design requirement.

There was also the question of NSA involvement. NIST said that the NSA would not be making the decision, they would. They said that the NSA would be providing comments and technical assistance, but that would be mostly through cryptanalysis. In other words, if the NSA could break a cipher, then that would certainly disqualify it, but the details (or even that an NSA break was the reason) might not become public.

There were a number of suggestions as to how to select a cipher. One suggestion was to simply rank the ciphers by speed, and then cryptanalyze them from fastest on down. The fastest cipher that passed a cryptanalysis would be the AES. Other people objected on the grounds that it would imply that speed is more important than security.

There were also people who believed there were political factors at work, as well. I remember an opinion that IBM was sure to win with their submission, MARS, because IBM had an inside track into government contracts and their history with DES. My opinion was that IBM was more likely to be at a disadvantage for the very same reasons. One major reason for the competition was to create trust for whatever the winner was. Thus, because of IBM's connections, they'd have to be a clear winner. They would actually be under a handicap.

There were also those who thought that national considerations would be a factor. Stated bluntly, "The US Government would never select a cipher written by foreigners[13]." And because for every conspiracy there is an equal and opposite counter-conspiracy, there were people who said that a weak cipher made by non-Americans would be the perfect thing, as that would mean the NSA could break it, and it wouldn't be their fault.

In 1999, NIST narrowed the field down to five ciphers. Of those, three of them were not only faster than Triple-DES, but faster than DES itself. Those three were also ciphers that had no intellectual property restrictions. Those two factors seemed to make the final five be a final three in popular thought. Those three were *Rijndael*[14], *Twofish*, and *Serpent*. The two remaining finalists were *MARS* and *RC6*.

Rijndael is the fastest of them. Serpent is slower, but more secure. Rijndael had the most innovative design, but in cryptography, innovative is not always a compliment. Serpent's creators were quick to point out that if speed were going to be the deciding criterion, then they could speed up Serpent by taking out its extra security margin and be as fast or faster than Rijndael. The Twofish designers pointed out that they could easily tune Twofish to be as fast as Rijndael or as secure as Serpent. Everyone had papers and reports showing how theirs was best by whatever metric you happened to want.

My opinion was that any of the three would be just fine. I have a great attraction to Rijndael. Rijndael is a very pretty cipher. Its internals are geometric and appealing. It also is fast. I disagreed with the sentiment of picking whatever was fastest and secure enough, but let's face it, fast is good. Twofish also appealed to me greatly, and I think if the decision had been mine, I would have chosen Twofish because it was a more traditional design. I would have wanted to pick Rijndael, but what if it has a huge flaw that we find fifteen years after the fact. In OpenPGP, we put Twofish into the standard, and to this day you can select it as a cipher in PGP software.

[13] I don't remember who said this, but that is as I remember someone saying it at a coffee break.

[14] You can't pronounce *Rijndael* correctly if you are not Flemish or Dutch; you just can't roll the R and the ij the right way. It is close enough for us non-Vlaams to pronounce it rain-doll or rhine-doll. I believe that rain-doll is the more Flemish pronunciation and rhine-doll the more Dutch, based on the way my Belgian and Dutch cryptographer friends pronounce it. I tend to rain-doll because that is how I heard its inventors say it.

NIST conducted a straw poll at the second AES conference in 1999. They asked people at the conference to rate the final five. If you read other accounts of the AES competition, you may read this poll called an election. It was not. NIST was clear at the time that they respected the opinion of the assembled cryptographers, but winning the poll would not imply selection, but would still be valued as the collected opinion of a lot of smart people. At the third AES conference in 2000, there was another straw poll.

In the end, NIST picked Rijndael to be the AES, which jibed with the way the polling went. Whatever went into the final decision, it made for an interesting result. Technically, Rijndael was the boldest design, and while there was no explicit emphasis given to speed, that was a large factor in everyone's favorable opinion. Politically, two Belgian cryptographers created the US standard. This has helped AES internationally, because it's hard to claim nationalism had anything to do with its selection. Since the selection followed the straw polls, whatever participation the NSA had in the process was almost certainly limited to cryptanalysis.

Most importantly, the AES competition truly advanced the art of cryptography. It stretched the art, giving requirements that were beyond the parameters of the then-usual cipher design. The five finalists are all good ciphers, and there is no reason to be afraid of using any of them. it also showed that security design can be done in the open, with participation open to the entire world. It showed that Kerckhoffs's Principle — the idea that you don't have to have secret design for security — works.

The AES selection worked so well that cryptographers have asked NIST to run another competition, this time for a new hash algorithm standard. NIST has held two preliminary conferences, one in October 2005 and the second in August 2006. It is being called the AHS, for Advanced Hash Standard [AHS]. The next few years will be very exciting.

The Crypto Wars

No one died in The Crypto Wars. No shots were fired, although a lot of ink was spilled. The term refers to debates, disagreements, between governments and everyday people in the 1980s and 1990s about the place of cryptography in society. A more adequate history of The Crypto Wars can be found in Steven Levy's book, *Crypto* [LEVY]. There are a number of factors that made The Crypto Wars inevitable:

- Governments had considered cryptography their domain. The advance of radio made cryptography necessary for them, and all aspects of cryptography and cryptanalysis had become strategic sciences. Moreover, the tale of how cryptography affected the Second World War was only just starting to be made public.
- The rise of what I characterized as computer cryptography created an interest in cryptography as a practical discipline in the civilian world. Public-key cryptography makes it possible to use cryptography in the civilian world because it solves the classical key distribution problem. Secondarily, the creation of a standard cipher like DES smoothed over other potential interoperability issues.
- Machine cryptography was still useful. The then-ubiquitous cipher machines used by governments all over the world were descendants of WWII cipher machines, made better with

more rotors and various other improvements, but they were still basically the same machines. Governments considered cryptography to be an art of war, and controlled it with the same laws and regulations that other tools of war were controlled [CryptoPolicy].

- The same sort of cloak-and-dagger work behind the scenes that happened in WWII was still going on. Machines were stolen, codebooks and key books were stolen. Machines were compromised with flaws colloquially called *backdoors* that enabled easier cryptanalysis.

  One of the major suppliers of cipher machines, the Swiss company *Crypto AG*, was involved in a scandal when the government of Iran figured out that their code machines were compromised. Apparently, the German and US governments had gotten them to put a backdoor in their machines [CRYPTOAG].

- Ironically, the combination of the US government creating DES and the publication of books about cryptography created even more interested in cryptography. Martin Hellman traces his interest in cryptography [Hellman] to Kahn's *The Codebreakers* [KAHN].

- Governments around the world acted in ways that, while understandable, were nonetheless high-handed, bullying, and peremptory. This didn't win them any friends in the civilian world.

I find it interesting that The Crypto Wars were created by the invention of the computer, which was created by the invention of the cipher machine. Looking back at it now, it seems like a long, slow-motion avalanche.

In the midst of The Crypto Wars, PGP software itself was created, and became a cause célèbre in the questions surrounding the place and classification of cryptography. The creator of the original PGP software, Phil Zimmermann, was investigated for breaking export regulations after a complaint to the US Government by RSA Data Security. After that investigation was dropped, the first PGP Incorporated was formed. Part of the approach to the export regime of the time was to take advantage of a sentence in the export regulations: *Printed material, including source code, is exempt from these regulations.*

That meant that if you printed the source code for cryptographic software, it could legally be exported, and if that printed source code was scanned back into a computer, the resultant software would be legally exported. PGP Incorporated and Network Associates used precisely this technique with the approval of the US Government to sell PGP software internationally without breaking the US export laws from 1998 to 2000.

Unsurprisingly, this is perhaps why The Crypto Wars ended with a fizzle. Cryptography is just mathematics, and free societies are uncomfortable in the long run with regulating multiplication as a dangerous technology. In 1999, France liberalized its cryptographic controls. Before, France had had the most severe restrictions on cryptography of any free country. Instantly, France became the most liberal free nation. In 2000, the US liberalized its export regulations, and there have been continued liberalizations throughout the world to this day.

While cryptography is still classified as a dual-use technology, meaning one that has both peaceful and military purposes, The Crypto Wars are over.

Chapter 4

The Basics of Cryptography

No one can build his security upon the nobleness of another person.
– Willa Cather

For the purposes of this introduction, we will look *only* at computer cryptography. Because this is to be a practical introduction, the way that human cryptography works and the way it doesn't aren't of value to us. Likewise, machine cryptography is no longer of practical value. We're going to jump directly into how computer cryptography and cryptographic protocols and systems are constructed and used.

We'll take a bottom-up approach. First I'll describe the components that make up a cryptographic system and then talk about how they're put together. We'll also look at the real-world protocols that PGP software uses and the larger systems that use those protocols.

Basic Components

There are a number of components that we use to make cryptographic systems.

Participants and Variables

Unlike most technologies, cryptography is ultimately about people who participate in some process. One of the things that continues to make it interesting to me is that it is about people and people talking. When we sketch out scenarios and problems, we have variables and those variables are themselves a language. When we use spacial math, we use the variables x, y, and z; x is horizontal, y is vertical, and z is up-and-down. If we add in angles, we use θ or φ or ψ for an angle. Various branches of math have not only used up the usual Roman letters, but also Greek letters, Hebrew letters, and even Gothic letters.

In cryptography, our main variables are participants, and often people. We don't use x and y or α and β, we use Alice and Bob. This approach is refreshing in many ways. It gets to the heart of the matter that this is a *practical* discipline. Alice and Bob are more to the point than A and B, but

that's what they are, just standard variables, about whom there is traditional gossip and speculation [ALICE]. They first appeared in one of Ron Rivest's early articles about the RSA cryptosystem, but I'm quite sure that Alice is the same Alice that Lewis Carroll wrote about, all grown up. Carroll would have liked it that she went into cryptography after growing up. We infrequently need a third participant, so there's less of a standard. Sometimes it is Carol, sometimes Charlie. On rare occasions we think, "Umm, Dave? Delia? Doris?" However, there are other participants with other rôles as well.

Eve is always the *eavesdropper*. Get it? It's more of what passes for humor in this discipline. Mallory is always the *man in the middle*who when talking to Alice, pretends to be Bob and when talking to Bob pretends to be Alice. If you search the Internet, you'll find earnest lists of other participants as varied, detailed, inconsistent and forgettable as Victorian codes of what it means when Bob gives Alice a rose of a particular color. Just remember Alice, Bob, and Eve. Mallory is also good to remember. If you see others, think of some pun on the name or an initial letter that is indicative of the rôle the person plays in the protocol.

Random Numbers

We use random numbers to create keys and other parts of our systems. Random numbers are also used in statistical systems, but cryptographic random numbers have to be different than those used for statistical purposes. Cryptographic random numbers must have the sort of properties that you'd expect a statistical system to have, but they also have to be unguessable. We typically call this property of unguessability *entropy*, and being computer people, we measure it in bits, which merely means powers of two. An easy way to think about scaling in bits is that ten bits is a factor of 1000. If you hear that something has thirty bits of entropy, the chance of guessing it is 1 in $1000 \times 1000 \times 1000$ or 1,000,000,000.

We spend a lot of effort making random number generators for cryptography. When you use PGP software, there are little things in the background, as well, looking at the way you use your computer. They take measurements of how you type and use the mouse and stir those into the random number generator pot to make sure there's no way to guess what comes out of it. These measurements are not the actual keystrokes and movements you make, but *timings* of those keystrokes and mouse movements. They also get scanned for regularities between them and that gets factored into the measurements. If you hold down a key and it repeats, the timings of the repeated characters will be the same. Those measurements will be rated much lower than if you type normally, and thus irregularly in timings.

Keys

Keys are the secrets of cryptography. The security of cryptography depends on the way keys are created, used, protected, and destroyed. In PGP software and other computer cryptographic systems, the keys are ultimately numbers, or if you prefer, strings of bits. There are three main ways to construct a key:

1. *Raw keys* are merely strings of bits coming from the random number generator. Most keys that we use are, in fact, raw keys.

2. *Derived keys* are keys that are produced from something else. For example, when we encrypt something with a passphrase, we do not use the passphrase directly, but derive the actual key from what you type as your passphrase.

3. *Structured keys* are a form of derived keys that we produce from some random numbers. RSA public keys, for example, need to have a certain mathematical structure and we start with raw random bits and find close numbers that have the appropriate mathematical structure.

No matter how they're created, keys are the — well, keys — to cryptographic security. As long as they're kept secret, you are as secure as the rest of the system allows. If they aren't kept secret, then you don't have any security.

Ciphers

Ciphers are the algorithms, the formulae, the recipes that we use to encrypt and decrypt. Ideally, they are a part of the cryptosystem that is completely known to everyone. With standard systems, such as those used in PGP software, they are. It is very tempting to make a secret cipher, but you should always mistrust a secret cipher. It is very easy to make a cipher that is good enough that you can't break it yourself. It is very hard to make a cipher that other people can't break. It is also very difficult to keep algorithms secret, particularly if you want to run them on a computer. There are many people who like to reverse-engineer systems, and the better you hide your algorithm the more tempting it is for them to find it. Many widely-used ciphers have been secrets, but none of them have remained secret. It is much better to spend the mental energy on something else.

As we discussed earlier, there two types of ciphers: public-key ciphers and symmetric ciphers. But why do we bother having these two types in active use? If public-key crypto has many advantages over symmetric-key cryptography, why do we even bother with symmetric systems any more?

The reasons are purely practical. Public-key cryptography not only uses keys that are often many times larger than symmetric keys, but it is many times slower. Public-key cryptography is at least 10,000 times slower than symmetric cryptography and thus for nothing more than speed of operation we use both types: public-key cryptography to send a symmetric key and then symmetric cryptography for speed and flexibility.

All ciphers have two measurements of size, one for the size of the key and and one for the amount of data the cipher encrypts at a time, called its *block size*. You'll hear both of these talked about.

Block Sizes

Public-key ciphers are the easiest to understand when it comes to their block size and key size. They encrypt a block of data that is the size of the key. So if you have a 2048-bit public key, it will encrypt chunks of data 2048 bits (or 256 bytes) at a time.

Symmetric-key ciphers have the most options. The block size of the cipher is unrelated to its key size. The ciphers we use today typically have a block size of 64 or 128 bits (eight or sixteen bytes). The AES has a block size of 128 bits, as do the ciphers developed as part of the AES competition

such as Twofish. The previous generation of ciphers, including Triple-DES, CAST, and IDEA have 64-bit blocks. There is also a class of ciphers that operate on a single character or sometimes even a single bit that are called *stream ciphers*. The RC4 cipher used in many SSL systems is an example of a stream cipher, and in fact one of the very few in active use. Stream ciphers may be considered a separate category of cipher from block ciphers because the fact that they encrypt a single item at a time leads to some interesting security characteristics.

Until quite recently, all ciphers were stream ciphers in one way or another. The Caesar cipher is a stream cipher. Enigma was a stream cipher, as were all the ciphers of that era. Similar to the way public-key ciphers made us invent a new term for the old way of doing things, the block ciphers caused us to create a new term for the old way of encrypting. Interestingly, the same thing happened with the creation of computer cryptography. Computers made it easy to encrypt in blocks, so we had to distinguish between stream and block ciphers.

Why would you want to encrypt in blocks rather than just one character at a time? As you might expect, block ciphers exist to address a problem in stream ciphers. Stream ciphers have the problem that if a cryptanalyst knows or can guess part of the plaintext message, it can help decrypt the messages and break the cipher. This is how Napoleon's codes were broken, and how Enigma was broken. It is also how the "WEP" (Wired Equivalent Privacy) cryptographic system for WiFi network connections was broken just a few years ago[1]. The more things change, the more they remain the same.

Bits of plaintext that you guess or know are colloquially called *cribs*, a term that dates back to the days of human cryptanalysts. It comes from the meaning of *crib* as a way to cheat at a test, not as a place to put a baby. A more common term today is *known plaintext* which isn't as poetic a term as crib, but is more straightforward. Known-plaintext attacks, ones in which the attacker knows not only the ciphertext but the plaintext that created it are the most powerful types of attacks there are.

Block ciphers are a way to address the danger of known plaintext because they take a block of plaintext and operate on it as one chunk. Mixing up all the bits in sixteen bytes of plaintext all at once makes it much harder to use known plaintext as a crib[2].

However, that isn't good enough. If all we did was encrypt blocks, we'd make the cryptanalyst's problem harder, but not hard enough. The cryptanalyst would still easily know that identical ciphertext blocks came from identical plaintext blocks. So to thwart this analysis, we do two things. First, we mix the output of one block with the inputs of the next. This strategy is called *chaining* and there are a number of ways to do chaining. Chaining has the effect of making connected blocks all different — or perhaps more precisely, if the ciphertext blocks happen to be the same, it's luck, not an indication of structure in the plaintext. Second, we start our encryption with one-blocks-worth of random data. This mechanism is called an *initialization vector* or IV. The initialization vector is not a secret. It is sent in plaintext, and thus does not need to be rigorously random the way a key does. On the other hand, it should be reasonably arbitrary as opposed to being something

[1]This isn't the only way that WEP was broken. WEP is an interesting case study not only because of the number of ways it was broken, but also because of how *easy* it would have been to prevent the breaks. All of the problems derive from using a stream cipher in an application that really calls for a block cipher. WEP is a good example of how strong cryptography can be used to make a weak system.

[2]There are others ways as well. One big one is never to reuse keys. Computers and public-key cryptography make this easier, as well.

like a counter. The initialization vector combined with chaining makes the output of a block cipher immune to cribs and known plaintexts (assuming, of course that the cipher itself is well-designed; there are plenty of ciphers that have fallen to known plaintext attacks). The result of these two additions to simple encryption is that plaintext encrypted this way has no exploitable patterns, even if it is encrypted multiple times (assuming you vary at least one of the key and IV).

Note, however, that some cryptosystems do not use chaining. For example, a number of disk encryption systems use the basic form of encryption. This makes them somewhat weaker than systems that go to the trouble of chaining. Even if they are using strong cryptography, they leak relationships between blocks of data. It is a small flaw, but a flaw nonetheless. All PGP software uses some sort of chaining. In PGP NetShare software, we have started using a new form of chaining called *EME mode* [EME]. EME mode is a "tweakable" mode we use with AES. It allows us to make disk blocks unlinkable no matter what data is in them.

Before the AES competition, almost all ciphers were created with 64-bit block sizes. When NIST started the AES competition, it asked that the block size of the new ciphers be 128 bits. The reason for 128 bits is that even with chaining, two ciphertext blocks could have the same value. Even if the blocks are the same by chance, it gives a cryptanalyst an interesting dollop of information that could be used to further attack the system. The probability of two blocks having the same ciphertext is related to a problem called *The Birthday Problem*.

If you have a room of people, what is the probability that two of them will have the same birthday? If you're looking for the number of people for which there a 50% chance of a duplicate, it is about at the square root of the total number of possible birthdays. This turns out to be $\sqrt{365}$ or about 23. In the case of a 64-bit block cipher, there will be even odds that two blocks that encrypt to the same value when you have encrypted about $\sqrt{2^{64}}$ blocks of data. That works out to be 2^{32} or about four billion blocks, or thirty-two billion bytes of data. That's a lot of data to encrypt with a single key and IV, but not as much as you'd think. DVDs in 2006 hold about 4.7 billion bytes. A very fast network can have that much data travel over it in a relatively few hours. This means that the higher-level protocols have to worry about stopping using one key and changing to another.

On the other hand, if you increase the block size to 128 bits, a *birthday attack* [3] has even odds of introducing a collision when you have encrypted $16 \times \sqrt{2^{128}}$ bytes of data, which works out to be some 295,147,905,179,352,825,856 bytes. This probability is remote enough that most of us don't really worry, much.

Families of Public-Key Ciphers

As I said earlier, public-key cryptography is based on the fact that there are mathematical functions that are easier to do than undo. There are two families of practical systems that we use today: the *factoring family* and the *logarithm family*.

The Factoring Family

The factoring family works by taking two prime numbers, p and q, and multiplying them together to get $n = pq$. Then you take n and do some math with it and your plaintext, giving the ciphertext

[3] I'll describe birthday attacks in detail when I talk about hash functions.

c. If someone sees the ciphertext, it is easy to undo that math if you know what p and q are, but difficult if you only know n. Since only the owner of the private key knows what p and q are, the system is secure. Of course, the two base primes, p and q, need to be of similar size. If you want a 1000-bit n, then you want p and q to each be 500 bits in size.

There are two members of the factoring family: RSA (named for its inventors, Ron Rivest, Adi Shamir, and Len Adleman) and Rabin (named for its inventor, Michael Rabin). RSA is the most widely-used public-key cipher today. The Rabin cryptosystem is very similar to RSA, but also uses squares and square roots.

Rabin has better mathematical underpinnings than RSA. We believe that breaking RSA is as hard as factoring. It is intuitive, and there is no reason to think otherwise—except that a proof is lacking. On the other hand, it has been proven that breaking a Rabin key is computationally as hard as factoring. However, there is a chosen-ciphertext attack in which the attacker persuades the keyholder to decrypt some data that is not an actual ciphertext, but a chosen string. The "decryption" of this garbage message can leak information about the primes p and q. For many years, this possibility was considered a fatal flaw of Rabin. Recently, we've developed a number of workarounds for this attack. Rabin is a viable alternative to RSA, but is still rarely used solely for network-effect[4] reasons. Despite being a fine cryptosystem, Rabin remains largely unused because it offers no compelling, practical advantages over RSA, not for any technical or cryptographic reasons. [RABIN]

For example, we at PGP Corporation could implement Rabin keys in PGP software, but there would be few reasons for users to create Rabin keys. Only other people who had software supporting Rabin could encrypt to them. The keys are not particularly smaller or particularly faster to use than RSA. Therefore, we stick to what we have out of a desire for simplicity more than anything else.

The Logarithm Family

There are a number of members of the logarithm family. The basic system is the Diffie-Hellman system, named for its inventors, Whitfield Diffie and Martin Hellman. Diffie-Hellman's security relies on it being easy to compute $m = g^x$ but hard to find out what x is. Diffie-Hellman proper is called a *key exchange* algorithm rather than a public-key encryption algorithm. Diffie-Hellman allows you and me to each think of a key, which is just a number, of course, and then do math that allows us to learn each other's key while keeping the keys secret to anything who observes the results of what we do. This process is also called *ephemeral* Diffie-Hellman, as none of the parameters are permanent.

Diffie-Hellman becomes a true public key algorithm in the Elgamal variant of Diffie-Hellman, named as you might expect for its inventor, Taher Elgamal[5]. The Elgamal cryptosystem takes the basics of Diffie-Hellman and creates out of it static key pairs that function the same way as RSA key pairs. You can encrypt and you can sign. The "Diffie-Hellman" keys that PGP software uses are more

[4]The *network effect* is also sometimes called the *fax effect*. In short, a telephone, fax machine, or cryptosystem is only widely useful if other people have them.

[5]If you do more study on public key algorithms, you will see Taher's surname sometimes spelled "El Gamal" (two words) or "ElGamal" (one word, but with an intra-capital). He has used these other spellings, but presently uses the "Elgamal" you see here because it's just easier when dealing with people who think that surnames must be one word and have one capital. It also spares him from the annoyance of getting mail addressed to "Taher L. Gamal."

properly called Elgamal keys. (The DSS signature scheme is a variant of Elgamal signatures, but we'll discuss that in more detail later.)

Note that all these systems use modular arithmetic for their calculations. Modular arithmetic is much less scary than it sounds. If you've ever used a clock or a car's odometer, you've used modular arithmetic. It is math that wraps. For example, if it is eleven o'clock now, and you have a three hour meeting, you're going to get out at two o'clock, not fourteen o'clock[6]. Similarly, you can't tell from the odometer if a clunker has two million miles on it or merely one million. In the case of the clock, we're doing modular math with a base of 12 (or 24), and with the odometer, it's modular with a base of 1,000,000.

For these public-key cryptosystems, we pick a prime number to be the the base. The reason we pick a prime number is so that when we repeat operations, there won't be short loops. Take the simplest of our examples, the twelve-hour clock. If you repeatedly add four, you get short loops. There are four different loops: $[12, 4, 8]$, $[1, 5, 9]$, $[2, 6, 10]$, and $[3, 7, 11]$. With a prime base, there will never be short loops. Why do we care about short loops? Because they could give an attacker an easier problem to solve.

Thus, if you look further in the math, you will see (for example) $g^x \bmod p$ rather than just g^x. I dropped all the mod p just to make it less confusing to look at[7].

Modular arithmetic, especially with a prime base, has another interesting mathematical property: you can do operations with integers that you wouldn't normally be able to do. For example, $\sqrt[5]{5}$ is not an integer, but $\sqrt[5]{5} \bmod 11$ is 1. Additionally, modular arithmetic in general keeps the numbers that we're working with to a fixed size. If the base p is a 1000-bit number, then any math we do will be with 1000-bit numbers. This property means that modular integers behave in mathematical ways like floating point numbers. In addition to adding and subtracting, we can take powers and roots and all those other things. It forms what we call a *finite field*, one with a finite number of numbers but one that permits us to do all the math for which we'd otherwise have to have an infinite number of numbers.

There are variants of the major public-key systems that work on quadratic curves, elliptic curves, and so on instead of the straight line of the integers. Usually, there is no particular reason to use these variants. They are interesting intellectually, but not practically or in an engineering sense. For example, you can do RSA with a quadratic curve, but the keys aren't smaller, the computer doesn't do less work, nor does the math take less memory. So why bother?

There is an exception, however, for the Diffie-Hellman family on a set of elliptic curves [ECC]. First and foremost, these keys are smaller. Second, the computer work needed to work them is often faster. This means that they are practical, and therefore interesting to us. The US government has stated that it wants to shift public-key cryptography to using elliptic curve keys over the next fifteen to twenty years because elliptic-curve keys are smaller and can be faster to use.

[6] Okay, smartypants, your three hour meeting starts at 2300 hours and gets out at 0200, rather than 2600 hours.

[7] Interestingly, the old Caesar cipher, the shift-by-N cipher is also a form of modular arithmetic, $c = p + k \pmod{26}$ for every ciphertext character c coming from a plaintext character p.

Key Sizes

Ciphers also have as a variable the size of their keys. As you might expect, there is a great difference between the key sizes used in public-key and symmetric algorithms. You no doubt have noticed that public keys come in sizes of thousands of bits while symmetric algorithms come in sizes of hundreds of bits. Why this difference, and how do they relate?

Let me start off with symmetric algorithms and say that 128 bits is good enough. In fact, it's more than good enough. Here is an illustration of what you would need to crack a single 128-bit key.

> Imagine the Earth covered with grass in a huge lawn. Also imagine that each blade of grass is a computer that can compute a billion keys per second. This cluster of computers would crack a 128-bit key on average in 1,000 years.

This description is a stunning testament to the power of exponentials. Our hypothetical key-cracker literally requires a spare planet covered with computers, and does not consider how you would power or cool those computers. I don't know about you, but I don't consider a single key in a millennium to be a practical attack. It might be nice to be so important that I'd have to worry about the eventuality, but perhaps not.

So why, then, do we bother with 256-bit keys in AES and others? The best answer is as a hedge against some *truly* science-fictional[8] technology that might get invented. When DES was developed, it was intended to last five years, and ended up being used for twenty. One of the design goals of the AES was that it be strong enough to last for perhaps fifty years.

With public-key ciphers, much bigger keys are needed to achieve *cryptographic balance* with symmetric ciphers. Just as a chain is only as strong as its weakest link, a cryptosystem is only as strong as its weakest algorithm. So if you want to have a public key in matching strength to a symmetric key, you have to have a much larger key. Table 4.1 gives NIST's recommendation for balancing the various parts of a crypto system. You can see from this table that they recommend a 3,000-bit RSA or DSA key to match a 128-bit symmetric key. You'll also note that for an elliptic curve public key, we only need a 256-bit public key.

Comparable Cryptographic Strengths						
	Size in Bits					
Symmetric Key	56	80	112	128	192	256
Hash function	160		256		384	512
MAC	64	160	256		384	512
RSA / DSA	512	1024	2048	3072	7680	15360
Elliptic Curve	160		224	256	384	512

Table 4.1: NIST's security equivalences in cryptographic algorithms

More importantly note that to match a 256-bit symmetric key in strength we need a 15,000-bit RSA or DSA key, but only a 512-bit elliptic-curve key. *This* is the reason the US government is

[8] I do not use the term *science-fictional* as a pejorative. I'm old enough to remember when going to the moon was science fiction and not nostalgia.

starting a push toward elliptic-curve cryptography. If you want to have cryptographic balance with the larger key sizes of AES, you need larger public keys, and with the integer systems, these keys become pretty large.

Unbreakable Ciphers or How Many Bits Are Enough?

PGP software, however, doesn't (yet) have elliptic-curve cryptography in it, and doesn't have 15,000-bit keys; it has 4096-bit keys. Should you care?

First of all, realize that a 128-bit symmetric key is good enough for the indefinite future. Current estimates [CSIZE] of how long keys should be usable say that in 2050, you should be moving away from 109-bit symmetric keys, 4047-bit RSA/DSA public keys, and 206-bit elliptic-curve public keys because an expenditure of about US$4.44 billion might be able to make a machine that can break a key per day. This estimate also assumes that DES was good until 1982, which is a very conservative estimate.

Deciding how many bits are enough implicitly asks the question, "How long do you need it secret?" If all you need to do is get a message to your stock broker to buy or sell before anyone else does, you don't need 128-bit keys. It is good to use them anyway, but you don't *need* them.

It is a delightful feature of AES that it also has 256-bit keys that are fast. AES-256 is only twenty percent slower than AES-128, and AES is usually replacing something slower than it. Consequently, many people use AES-256 for reasons that have nothing to do with security and everything to do with marketing.

This is the sort of trade-off that we have to do all the time. There's no reason *not* to use AES-256. It's a fine algorithm, and everyone thinks it gives a full 256 bits of security. It is also only a bit slower than AES-128. Therefore, there is a great push toward it. People use 256-bit keys because all the cool kids are using them. Of course, this means that if you actually need that extra twenty percent of performance, then you want AES-128 and sometimes someone made that decision for you. Sad, isn't it?

Perhaps, but it's not as bad as abusing other ciphers. Before AES, there were a number of ciphers that were designed with variable-length key sizes. Most were intended to be used with 128-bit keys, but supported larger ones. It wasn't until the AES competition that ciphers were designed with large keys as a goal, rather than as a feature. If, for some reason, AES makes you uncomfortable, the AES competition produced other ciphers that are also fine ciphers. PGP software supports one of them, Twofish [TWOFISH], with 256-bit keys. Use that; it's a great cipher.

When we select a public-key size, there are other considerations we take into account as well. The main one is speed. Today, just about everyone thinks that 1024 bits is too small for public keys. (The [CSIZE] report says you should have ditched those in mid-2001.) If you're using long keys, you may gnash your teeth if you're using a small device. I have a BlackBerry running PGP, and a 3,000-bit key takes a while to use. The poor little thing is internally a 40MHz 80386. That's what we might have used on a desktop machine when PGP was first written, back in 1991. If you are using a 4096-bit key on your mobile phone, you may decide to change it to 2048 bits simply because you're tired of waiting for the decryption with a longer key. In contrast, you'd never even notice the difference on even a five-year-old laptop.

Certainly, the small, mobile devices we have now will continue to get faster. As always happens, the system that was in a desktop last decade and a phone this decade will find new life in something like a door lock or a light bulb next decade. You'll want those to be secure, but not take even ten seconds to turn on the light.

Security is context-dependent and more bits are not always better; you need to consider what you're protecting because there is only one type of perfectly secure system, which I'll discuss next.

One-Time Pads, the Truly Unbreakable Encryption

One-time pads are the only *completely* secure cryptosystem there is. Beyond *my* specialized use of the word *perfect*, one-time pads are *truly perfect*, perhaps I should say *pluperfect*. They were perhaps most famously used by the British in WWII [MARKS] and most infamously by the Russians both before and after [VENONA].

One-time pads are easy to understand, and easy, if tedious, to use. In Figure 4.2, I've constructed a small one-page one-time pad as an example. This particular one uses the letters of the alphabet and a space for a total of 27 characters. The pad has 100 total cells. Imagine a pad of paper with each page printed like our example.

Page #1 of 200, use once and destroy									
17	23	14	0	7	10	22	9	6	18
3	16	11	15	17	16	21	5	8	11
6	12	23	1	18	6	19	14	16	11
22	6	13	25	1	0	4	11	24	5
17	10	23	23	11	15	10	12	2	6
24	19	21	26	2	5	15	10	7	9
26	2	22	17	10	0	11	8	3	6
18	19	19	26	1	11	14	24	12	19
18	20	7	7	4	8	1	11	25	0
9	3	14	16	4	17	6	1	16	1

Table 4.2: Sample One-Time Pad

To encrypt, we do the following:

1. Take the first plaintext letter in the message.
2. Shift the letter by the number in the next cell of the pad. Shifting A by two gives us C. Note that a space is the 27th letter. Mathematically, $ciphertext_i = plaintext_i + pad_i \pmod{27}$.
3. Repeat this process for all the letters in the message. If the message is larger than the pad, use the next page of the pad. Repeat as necessary.
4. Destroy the used pages of the pad. Eating it works. Burning it is better. (If you're going to do both, remember that the order is important and makes the difference between being painful and merely unappetizing.)

I'll leave the decryption process as an exercise for you. Here is the test ciphertext of:
DKJ PBVBNWCITAVP TZKGXHAYUGRPNMDBRPGVLMMVAOWDCJOQSBSNN.

Why is this pluperfect encryption? Note that the key for the encryption is the entirety of the pad. Each cell of the pad has a random number in it. We add that random number to one and only one letter of the message. Assuming the random numbers are really random, the *entropy* of the pad cell, its *unguessability*, is equal to the information in the message letter. In the whole of the message, the entropy of the key is equal to the information of the message. Therefore, there is no way a cryptanalyst can use patterns, statistics, or anything else to pry any information or meaning out of the message. Even if I taunt you, the cryptanalyst, with the knowledge that the first word of the message is indeed a three-letter word, there is nothing to give away which particular word it is. The? His? And? Why?

You know that the message is 55 characters long, but that fact isn't a lot to go on, and evil person that I am, I might have stuck in a few extra spaces on the end. As I created this example, I considered adding in a STOP character to the alphabet, like those used in telegrams of old. I could then end a sentence with STOP and the whole message with STOP STOP. Then I could pad it out to an even 100 characters with NYAH NYAH NYAH or anything else I wanted to.

This is the beauty of one-time pads. They're beyond perfect. They're pretty. They're dazzling. And although they're tedious [TEDIUM], they're still fun. They're seductive.

The Seduction of the One-Time Pad

It's not easy being pretty. There are a lot of operational considerations you have to take care of when you use a one-time pad.

- The pads have to be random. Really random, not pseudo-random. If they are not really random then the cryptanalyst can use that non-randomness to start prying at the message. While doing research for this section, I found a web page that did some one-time pad encryption similar to my example above, but it used a pseudo-random generator that you seed with something you type in. When Alice does that, she loses the perfect security. The security of the system is now the security of that seed. The seed *is* the key; if Eve can guess the seed, then Eve can decrypt the message.
- The pads have to be kept secret. Both Alice and Bob have to make sure that no one else can get to them. If Eve bribes Bob's housekeeper, Hattie, or Alice's bodyguard, Justin, then she can trivially decrypt the messages. This security requirement also applies to the used pages. My suggestion about both burning and eating them was not *completely* facetious. In WWII, Leo Marks had one-time pads printed on silk for easy burning and not much ash. It's not unheard of for sensitive pads to be printed on flash paper for extra-easy destruction.
- Alice and Bob must have at least as many pad characters as characters in the messages they send. They need to have a reservoir of pad material in advance of any of the messages. This means that they need to have some form of trusted courier to distribute new pads in advance of their being needed. Presumably, Alice's bodyguard, Justin[9], will suffice for this. Note,

[9] I don't know about you, but I've always thought Alice's bodyguard ought to be named Justin Case.

however, that this at least doubles the communications they have to do, because the pads are bigger than the totality of their messages.

On the Internet, this process of keeping enough pad material is even more difficult, because it's hard to know how to securely send a one-time pad over the network without having first sent (and then expending) a previous pad. Compression won't help, because you can't compress random numbers.

- Should they run out of pads, Alice and Bob have to stop sending messages until they get new pads. As tempting as it might be to reuse one, they have to resist the temptation. In the famous [VENONA] case, the Russians reused their one-time pads when they ran out. The American NSA used this error to decrypt their messages. It wasn't easy; it took decades, and cryptanalysts literally went mad while spending their career decrypting these messages. Nevertheless, it can be done, and why go to all this trouble for cryptographic security if you're not going to follow through with proper security procedures?

These additional issues are why I use the word *seduction* to describe the attraction of one-time pads. It's true they offer perfect cryptographic security. The problem is that perfection can seduce people into thinking that their *operational security* is perfect, too. A chain is only as strong as its weakest link, and so is system security. The total security of the system is the least of all of its parts, not the most.

If instead of a one-time pad, Alice and Bob use a cryptosystem like PGP encryption, they are trading off cryptographic perfection for operational simplicity. If the difference in chance that a pad will fall into Eve's hands as opposed to public keys falling into Eve's hands is greater than 2^{128}, then it is reasonable to think that public keys are more secure than one-time pads.

I don't know about Alice and Bob, but I know about me. I'm smart, but slightly sloppy. I have a cluttered office and sometimes I get distracted. When I created PGP Universal, I built upon the cryptography that Phil Zimmermann and the PGP team had created and added automated operational security. For me, this is an improvement in overall security because I no longer have to think about encrypting.

I'm not immune to the seduction of the one-time pad. It would be really cool to somehow get that perfection into a practical system. But I'm also reminded of a common theme in fiction: the protagonist is dazzled by the beauty and perfection of someone while ignoring the plainer, but more devoted and suitable, love interest nearby. Usually, the story is a comedy and the protagonist comes to his or her senses at the end. The only tragic example I can think of is Shaw's *Pygmalion*, in which Eliza tells Higgins off. That ending was so jarring that the story is much better known as *My Fair Lady*, in which they live happily until the sequel. One-time pads are so very pretty, but also very high-maintenance. There are places they've been used or are being used well. But the reason the world at large has moved to public-key ciphers and advanced symmetric ciphers is that 128 bits is enough, unless your adversary has a planetful of computers. If that possibility still worries you, 256 bits is even better—modulo truly science-fictional technology.

Hash Functions

Hash functions[10] are an important part of cryptography. They are the workhorses that we cryptographers use and abuse for all sorts of things, and yet we understand them least of all the cryptographic primitives.

A hash function takes a variable-length input string and creates a fixed-length output. That "hash" of the input is a shortcut, not unlike a person's initials. You can refer to the input string by its hash. The fact that it is a fixed-length string allows us to easily use the hash value as a referrer to the actual string itself.

Because a hash function takes a long string and reduces it to a short one, it is inevitable that there will be two strings that hash to the same value, or *collide* in cryptographer-speak. For example, the names Jon Callas and Jane Cannoy collide with initials to the hash of JC. Collisions are important in the understanding of hash functions, and we'll talk more about them in a bit.

Although initials are an easy way to describe the basic concept, initials make a bad hash function for cryptographic purposes. A cryptographic hash function has a number of other properties that make it useful cryptographically.

- It should be *hard* to reverse a hash function. Knowing the hash, there should be no good way to find the input string that generated it. Given that (typically) hash functions lose data, this is a relatively easy property to create. The same property is also true for initials: knowing JC and nothing else, there is no good way to get to my name.
- Given a hash value, it should be *hard* to identify a possible source string. This property is one that initials lack. It is very easy to look at a set of initials and know if a name matches it. With a cryptographic hash function, however, we want the relationship between a source and a result to be as opaque as possible.
- Given one source string, it should be *hard* to find a second string that collides with its hash. It should be especially hard to change a string usefully and get a collision. In an extreme case, it should be hard to change "I agree to pay $100" to "I agree to pay $500" and have that collide. Note that the difference between the two strings is only a single bit.
- It should also be *hard* to find two strings that collide in their hash values.

These requirements give us very flexible functions that are used for lots of different things. Here are some examples:

- When you type a passphrase into PGP software, we use a hash function to turn it into a key to use with a cipher. The core of that conversion is a hash function, usually used over and over again to slow down an attacker trying a brute force attack on the passphrase.
- PGP software's random number generator continually updates itself by feeding in information about your mouse movements and keyboard typing. Although these observations are not at all deterministic, they are also not uniformly random. We use hash functions to smooth out the unevenness in the observations.

[10] Hash functions are also sometimes called *message digest* functions.

- Random number generators themselves often use hash functions to produce their output. The one in PGP software does.
- File integrity systems use hash functions as quick checks on the files. For example, you can keep a list of the hashes of the files on your computer, and you can see if that file has changed by comparing the hash of the file on disk to the one in the database. Software distribution sites also often list the hash value of the distributed file so that people who want to see if they have the right file can compute and compare hashes.
- Complex cryptographic systems that create data integrity use hash functions as a component. We'll talk more about them later.

Note that for almost all of these uses, there's an assumption that there won't be collisions. If two passphrases collide in their hash, either can decrypt a file. If two software packages hash to the same value, then one can be mistaken for the other.

Commonly Used Hash Functions

Table 4.3 lists some commonly used hash functions, especially the ones we presently use in PGP software.

Difficulties with Hash Functions

Presently (mid-2006), we know the suite of hash functions we have been using is not *perfect*, and some of them are quite imperfect. These problems came to light in the summer of 2004 when Xiaoyung Wang announced that she and her team produced collisions in a number of hash functions [WANG04]. Adi Shamir, the "S" in the RSA algorithm, said at the time, "Last week, I thought that hash functions were the component we understood best. Now I see that they are the component we understand least." In early 2005, Xiaoyung's attacks were extended to SHA-1, which had survived her first work [WANG05].

We are still coping with these problems, all of which revolve around hash function collisions, two strings producing the same hash value. One of the axioms of the branch of mathematics called combinatorics is called the *Pigeonhole Principle*. At its simplest, the Pigeonhole Principle states that if you have thirteen pigeons and only twelve pigeonholes, then at least one hole must contain at least two pigeons. Pretty obvious, isn't it? That's why it's an axiom.

If you apply this principle to hash functions, consider sixteen-byte hashes. Also consider the entire set of seventeen-byte strings. According to the Pigeonhole Principle, there are going to be at least two original strings that produce the same sixteen-byte hash. As a matter of fact, there have to be a whole lot of them. The collisions are the equivalent of pigeons lumping themselves together. If the collisions are evenly distributed (and thus the hash function *perfect*), then there will be 256 collisions per hash value, and according to the Pigeonhole Principle, there has to be at least one hash with at least 256 collisions.

Finding a collision ought to be no better than guessing, but how hard is that? Answering that question raises another interesting mathematical problem called *The Birthday Problem*, which we

Name	***Size***	***Description***
MD5	128 bits	MD5 was the sole hash function that PGP software used prior to PGP 5.0. Weaknesses in MD5 first showed up in 1996. MD5 is itself an improvement on MD4, which was never used in PGP software and was the first common hash function to be fully broken.
SHA-1	160 bits	SHA-1 appeared in PGP 5.0, and also in OpenPGP. SHA-1 is an improvement on MD5 that was created by NIST to be wider and also to correct problems in MD5.
RIPE-MD/160	160 bits	RIPE-MD/160 is a hash function similar to SHA-1. RIPE-MD/160 was created to be an improvement over MD5. However, it was created by the European *Réseaux IP Européens* (RIPE) organization rather than the US NIST. We expect it has similar security characteristics to SHA-1.
SHA-256	256 bits	SHA-256 is one of a new family of hashes created by the US NIST that are collectively called the "SHA-2" family. It has different internal structure, but comes from the same basic construction as the other hash functions in this table.
SHA-512	512 bits	This is another member of the "SHA-2" family, along with SHA-256.
SHA-384	384 bits	SHA-384 is a variant of SHA-512 that has a smaller output. In general, SHA-384 is not used, because it has no advantages over SHA-512 except for the hash size. It runs at the same speed as SHA-512, so usually if we need something stronger than SHA-256, we go directly to SHA-512. There is also a SHA-224 which is a similar truncation of SHA-256.

Table 4.3: Commonly Used Hash Functions

first saw when talking about block sizes. The probability that a given person has the same birthday as Alice is about 1/365 [11]. But if you have a room full of people, what is the chance that there will be a collision on their birthday? Specifically, with how many people are there even odds that there will be two people in the room who share the same birthday?

The general case answer to this question is the same as finding collisions in a hash function. We can think of a birthday as yet another hash function with perhaps better properties than initials, but still nowhere near perfect. Nonetheless, birthdays are fairly randomly distributed[12]. For birthdays, it turns out that the odds of a birthday collision are even at about 23 people. In the general case, the odds are even at about the square root of the number of options.

I'm sure you noticed my use of the weasel-word "about." This is because the answer isn't exactly the square root, but close to it. In the general case, the chance of collisions is

[11] We're going to ignore February 29.

[12] It turns out that more people are born in August than other months and there's even a little more of a spike in the third week of August which says something about festivities in December. It also turns out that more people are born on a Tuesday than any other day, but that is caused by different effects.

$Prob(pigeons, holes) = 1 - \frac{holes!}{(holes - pigeons)! \cdot holes^{pigeons}}$. Sparing you lots of math [Birthday], if you solve this problem for the the number of pigeons that will give you the probability of $\frac{1}{2}$ of two of them in the same hole, it works out to be about $1.2\sqrt{holes}$, which is close enough to $\sqrt{holes}$ that we typically just use that answer, especially because we're going to be dealing with very large numbers.

So, if we have an n-bit hash function, there are even odds of a collision when we have $2^{\frac{n}{2}}$ strings we've hashed. Thus, we say that a 160-bit hash function should have 80 bits of security. 2^{80} is a very large number. It's about twice Avogadro's Number, which is the number of molecules in a mole, or to put it in convenient terms, the number of molecules in a rounded tablespoon of water. It's a big number. When Wang came to Crypto 2004 with [WANG04] in hand, it shook cryptographers up. She didn't have a paper that showed how to create collisions, she merely had a lot of them. As you can see, because collisions are supposed to be hard to find, merely *possessing* a handful of collisions on each of a handful of 128-bit hash functions means that something is up. For cryptographers, the main question was, "What does she know that we don't?" Six months later, her techniques were extended to attack the prime 160-bit hash function.

Here is a summary of what we've learned in the last two years:

- Wang is an excellent cryptanalyst. She doesn't have any fundamental mathematical insights that other mathematicians don't have; she's merely the world's best hash function cryptanalyst by leaps and bounds.
- Some other theoretical work that wasn't particularly practical is getting a lot of thought. For example, a few months before [WANG04], John Kelsey and Bruce Schneier showed in [KSHASH04] that when looking for a SHA-1 collision of a given string, you could do it in 2^{106} work instead of 2^{160} but you need to have messages 2^{60} long to be able to do so. Before Wang showed flaws in how we were doing things, this was interesting but not practical. Now some of us wonder if this impractical flaw is an indication of a structural problem. We don't know, yet.
- There are a number of proposals on how to modify existing functions to withstand Wang's attacks. They're all very good, but the obvious follow-on question is, "What new attack will happen *next* year, that this fix doesn't account for?" Of course, this question is unanswerable. We just can't protect against unknown attacks. However, a number of these proposed solutions are practical to implement. Simple techniques like doubling every byte as you hash (instead of hashing ABC, hash AABBCC) or inserting a zero byte after every four bytes [SY05] or adding in some random data at the front of the data to be hashed protect against these known problems.
- We're starting to get an idea of how to design hash functions better. In October 2005, NIST hosted a workshop on hash functions, and cryptographers are starting to get a better idea of how to make good hash functions. A second workshop is planned for August 2006. There is also growing support for a competition similar to the AES competition to produce new hash functions.
- There are also good ideas on how to proceed from an engineering standpoint. At PGP Corporation, we have been leading this initiative.

At PGP Corporation, we started migrating away from MD5 back in 1997. PGP 5.0 started a gentle migration away from MD5 toward SHA-1, keeping MD5 solely for backwards compatibility. PGP 8.0.3 introduced support for reading but not generating SHA-256, SHA-384, and SHA-512. PGP 9.0 started a gentle migration from SHA-1 to SHA-256.

Data Integrity Functions: MACs and Signatures

Our next building blocks are those that let you know that a piece of data is intact — that it is the data you expect, the whole data, and nothing but the data. Like encryption, there are two basic ways to do data integrity, with symmetric or asymmetric keys. These are called MACs, for *Message Authentication Codes* and Digital Signatures.

You can make a MAC with either a cipher or a hash function; the latter are called HMACs. Digital signatures are much more powerful than MACs, but MACs are faster. Since a MAC uses a symmetric key, anyone who knows the key can rewrite the data and the MAC that goes with it. They work very well between two parties, but less well when they're needed among many parties. In contrast, a digital signature is made with the private half of a keypair, and anyone who has the public key can verify it. This property allows digital signatures to be broadcast with one signer and many verifiers. They're used for checking that anything from messages to software are not only intact, but come from a designated source. This is very useful in communications as well as in storage. MACs in protocols like SSL keep an attacker from undetectably changing the flow of bits or adding in something or deleting something.

The common algorithms that we use for digital signatures are RSA and DSA. RSA can sign as well as encrypt. DSA is interesting in that it is a signature-only algorithm. There is also an elliptic-curve variant of DSA known as ECDSA. It is also possible to create digital signatures with the Elgamal algorithm, but this isn't typically done. At one time, the OpenPGP standard allowed Elgamal signatures, but they have since been removed.

If you do more research that goes into the actual mathematics behind digital signatures, you'll see some descriptions that talk about signatures being the same thing as encrypting, but you encrypt to the private key rather than the public key. This is true for RSA, but not for DSA nor Elgamal. The reason for this is that Elgamal signatures are somewhat tetchy to create and use. DSA signatures are actually a variant of Elgamal signatures created to be less problematic, but with the added advantage that they are much shorter than Elgamal or RSA signatures.

No matter what algorithm you use to create a signature, the data itself isn't signed, but rather the signature is on a hash of the data. The reason that we do this is two-fold. First, as you remember from encryption, a public key algorithm operates on a chunk of data, and you have to handle the case where the data is larger than the key (which is the vast majority of cases), by doing it a number of times. You would end up with a signature that is as large as the data you signed, and would have to chain the pieces together. Secondly, this would be very slow. So to get a small object quickly, we hash the document and sign that. So long as the hash function behaves well, this is a reasonable thing to do.

Note that this is also a reason why cryptographers are so concerned about the strength and security of hash functions. Digital signatures are actually a composite of raw signing and hashing. If there are problems in either then there are problems in digital signatures.

Digital signatures are important because they provide data integrity as well as cryptographic control on the data integrity. Hash functions alone can tell you if data has changed, but you have to keep a separate list of the hashes of all the data you want to track. With a digital signature, if you are the verifier, you need to keep the data and its signature. You also have to have the public key of the signer, but you only need one of those per signer. As the signer, you need to keep the signing private key protected, but anything you signed does not need special care. Interestingly, the signer doesn't even have to keep the private key. It is possible to sign something and then destroy the private part of the signing key. The signature can always be verified even if the signing part is lost to the mists.

Certificates

Public keys are like telephone numbers; you need to have the one of the person you want to work with. They are also like telephone numbers in that there is a lot of metadata about them that we have to track. If you are unfamiliar with the term metadata, it is merely data about data. Consider a book; there is not only the actual contents of the book, but there's also its title, its author, what edition and printing it is, whether it is soft or hard cover, its ISBN number, the subject, genre, and so on. These are all metadata about the book[13]. Now consider a telephone number; the metadata about it include the name of the person it will ring, whether it's a home, office, mobile, or fax number, and so one. When we consider keys, there is also metadata about them. Like the telephone number, there's the name of the person it refers to. But there are also other metadata about a public key, such as how is it to be used — for encrypting, signing, both? When was it created? Is it not supposed to be used after a certain date?

We call the blob of data that holds a key and its metadata a *certificate*. We also call them *keys*, particularly when they're intended for use with OpenPGP. The convention of calling PGP thingies *keys* rather than *certificates* came from Whitfield Diffie. The word "key" is a nicer word than "certificate." It sounds better, it is easier to say, it is less jargon than "certificate " is, even if not being entirely accurate (and that I will discuss later). Colloquially, OpenPGP certificates are called keys. I tend to use the word certificate, however, because I want to make it clear that an OpenPGP key is a certificate.

Certificates are nothing more than the data structures that hold a key and metadata about that key. They may come in one format or another, but we have them because keys alone aren't all that useful. We want to work with keys and their metadata, especially because that metadata will contain important data about how that key is to be used. There have been many attempts over the years to get rid of certificates, some successful, some not. Certificates continue to be the dominant way to hold keys because rarely do you need a key but no metadata.

[13]We can debate some of the finer points. Is the book's title really metadata, or is it the book itself? I can see the other side of the argument. Someone else might consider the index and table of contents metadata, but I don't. They seem to me to be part of the book itself. Nonetheless, once you consider that there is the book itself and things about the book that nonetheless go with the book, you understand metadata.

Why Certificates?

Let us consider a key (and by this, I mean a bare key) we have for Alice. Cryptographers are wary creatures and we worry about the most obvious things. How does Bob *know* this is Alice's key? If Alice gave it to Bob, it's pretty straightforward, just as if Alice gave Bob her telephone number. But this doesn't scale. With telephones, we have directories. We look up people's numbers in the phone book. These directories create a binding between someone's name and their phone number. If we wanted to have an analogue to a telephone book for keys, we would want to have some assurance that it is accurate.

Digital signatures allow us to do something extremely clever – why not take all that metadata about Alice as well as her key, and sign it? Then you can verify the signature and use the data integrity provided by the signature to have some indication that the mapping of that key and all the metadata is *valid*. We use that word, valid, intentionally. Just as in logic and reasoning to differentiate between *valid* (meaning the the processes are correct) and true (meaning that the facts are correct). You can have a valid line of reasoning that is nonetheless false. You can also have a valid certificate that is not accurate. For example, if Alice changes her name, a certificate with her key would no longer be accurate, but it would still be valid.

There are two major syntaxes that we use to create certificates. These are nothing more than data formats, similar to the way that there are different formats for pictures. We'll discuss them more as we go on, but these two formats are OpenPGP and X.509. Today, PGP software can work with certificates in either format and oftentimes will take the actual public keys and dress them in whatever uniform is best for the purpose at hand.

A certificate can be signed by any key. Usefully, the certificate might be signed by the key it holds. We call these *self-signed certificates*. For example, if Alice gives Bob her key in a self-signed certificate, that's all he needs. At any time, he can verify that the information in the certificate hasn't changed. Also usefully, that certificate can be signed by some other key than Alice's. Perhaps Bob signed it himself, but it's also likely that some third party, Charlie, has signed that certificate. In the OpenPGP world, we call these third-parties *Trusted Introducers* but in the X.509 world, Charlie might also be known as a *Certificate Authority* or CA.

Trust and Authority

Trust is a funny word. It means so many things in so many contexts. In this case we're going to use a narrow, strict definition. *Trust* is the mechanism that use to decide that a certificate is valid. A *Trust model* is a broad scheme that we use for trust. I know this sounds confusing, but it makes more sense than you'd think. Let's look the basics of trust models.

Direct Trust

Direct trust is the most straightforward. Bob trusts that a certificate is Alice's because Alice gave it to him. It is the best trust model there is, so simple that I already described it to you without giving it a name. It is not only the simplest trust model, but at the end of the day it is the most – well, trustworthy!

People use direct trust all the time by doing things like putting an OpenPGP key fingerprint (which is itself a hash of an OpenPGP key) on their emails or business card. I do this, myself; if you get my business card from me, there's my key fingerprint. You can then use this to know that my key is actually my key — you can trust that the certificate is valid.

As I said earlier, the only problem with direct trust is that it doesn't scale very well to something the size of the Internet. That doesn't mean it's not useful, it just means it doesn't scale.

Hierarchical Trust

Hierarchical trust is also straightforward. In hierarchical trust, you trust that a certificate is valid because it was signed by someone whom you believe to be accurate, as in the example I gave in which Bob considers Alice's certificate to be valid because Charlie signed it. Bob trusts Charlie to be an authority on accurate signing.

There are a good number of companies that make it their business to be Charlies. GeoTrust and VeriSign, to name two, are widely trusted Certificate Authorities, and they are trusted because of the practices they follow in creating certificates. Part of these practices are that they have a relatively few number of keys that extend this trust. These keys are themselves held in *Root Certificates* (which are self-signed certificates). The key in this root certificate signs a key in another certificate. This can extend some number of times before we get to the actual certificate we're interested in.

To verify that certificate, we trace a chain of certificates. Alice's certificate is signed by a certificate that is signed by a certificate that is signed by the certificate that Jack built. Many large corporations, governments, and so on have their own hierarchies that they create and maintain.

X.509 certificates of the sort that we use for SSL connections on the web use hierarchical trust. If you go to Amazon.com, that web server has a certificate that was signed in a hierarchy that traces up to a root certificate that we trust. For these commercial certificate authorities, they typically use a depth of two or three.

Ah, I can hear you ask, "But how do we trust that root certificate?" The answer is: direct trust. Built into your web browser is a set of root certificates. You consider them valid because they are part of the software you installed.

Hierarchical trust is straightforward, but has an obvious risk. If the authority makes a mistake, the effect of that mistake is great. In a notorious example of this problem, Microsoft uses a hierarchical trust system for its code-signing system, Authenticode. That hierarchy is maintained by VeriSign. A few years ago, some miscreants convinced VeriSign that they were Microsoft employees, but they were not. Fortunately, the scam was caught quickly. Had they gotten away with the improperly-issed certificates, whoever ended up with those certificates would have been able to sign software with a Microsoft key, and as far as the system works, that bogus software would have been Microsoft software.

Cumulative Trust

The drawback of hierarchical trust — that a hierarchy is brittle — leads us to the last major trust model, that of cumulative trust. Cumulative trust takes a number of factors into consideration and decides if a certificate is valid based upon these factors.

The most widely used cumulative trust system is the PGP *Web of Trust*. In the Web of Trust, Bob can assign various *trusted introducers* (who are actually certificate authorities, even their names are Charlie and Dale) a ranking in points. If there are enough points, Bob considers the certificate valid. In OpenPGP, we have three rankings for introducers, *untrusted* (zero points), *partially trusted* (one point), and *fully trusted* (two points). If there are enough certifications to reach two points[14], then the certificate is considered valid.

You can also have trust cascade from one introducer to another. For example, Bob might consider Zelda's certificate valid because Alice signed it, and he both trusts Alice and considers her certificate valid because both Charlie and Dale certified Alice.

The cryptographer Ueli Maurer [MAURER] developed an extension to the PGP Web of Trust that allows for a continuous scale of trust assignments. In his model the usual PGP scale would be zero, one-half, and one with validity granted when enough trust accumulates to get to one. However, he permits any fraction to be assigned to an introducer. You can have one introducer given a value of 0.9, and another of 0.15.

Cumulative trust can encompass both direct trust and hierarchical trust. It is easy to set up in a cumulative trust system both direct trust and hierarchies of trust.

Hybrids of the Trust Models

None of the basic trust models are widely used in their pure forms. There are too many advantages of each. Even in worlds that use strict hierarchies, there are authorities called *Bridge CAs*. These are certificate authorities that function as introducers of one hierarchy to another.

In the PGP world, we also use hybrids. The PGP Global Directory is a mini-hierarchy within the larger web of trust. If Alice submits her key to the PGP Global Directory, the Global Directory sends an email to all the email addresses she has put on her key. If she responds to those emails, those addresses are listed in the Global Directory. The Global Directory also sends follow-up emails every six months. It then certifies her key every two weeks.

Additionally, PGP provides for domain-level trust and directories. If a domain sets up an OpenPGP key server at the host `keys.domain`, for example, `keys.pgp.com`, a number of OpenPGP systems (including PGP Desktop, PGP Universal, and Hushmail) will automatically discover keys in that directory. It considers a domain-level key in that directory to be a trusted introducer for that domain. For example, at `keys.pgp.com` we have a such a directory and in it there is a key for the domain `pgp.com`. That key has signed mine, and therefore you will automatically consider my key to be valid. Note that this is a combination of direct trust of an introducer for that domain with the other parts of the web of trust, and that creates its own mini-hierarchy.

Certificate Dialects and Gory Details

As I said earlier, there are two major data syntaxes for certificates, X.509 and OpenPGP. They're the same yet different. Many products, such as PGP, can work with both certificate types. As time

[14] In PGP's software, there is also a setting for having a threshold of one point.

goes on, the differences between them will be less and less relevant. Analogously, there are also multiple formats for pictures and reasons they all exist. But when you put together a web page or a photo album, the differences mostly fade away. Additionally, each format grows a little closer to the other one all the time. If the people who *use* certificates like something than one has, the other usually acquires it. But let's look at a number of the differences in how X.509 and OpenPGP are used.

Certificates and Certification

X.509 certificates have a simple, basic structure that I described above. They contain a key, information about the key, and a signature that is a statement that they go together. OpenPGP certificates are more complex: they may hold more that one key, more than one clump of information, and more than one signature. Despite this, the formats are not incommensurate. You can express the same thing that an OpenPGP certificate does with a group of X.509 certificates.

For example, the SSL certificate that www.pgp.com uses contains a public key, the hostname of the web site (www.pgp.com), some other information (for example, that the signer does not recognize certificates made with the key in this certificate), and a signature made by the certificate authority, GeoTrust.

The OpenPGP certificate that I use for email also has a public key. It has information (the email address `jon@pgp.com`), and a signature binding those together. This signature is a self-signature, one made by the public key itself. There is also another signature, one made by the PGP Global Directory (<`http://keyserver.pgp.com/`>). There is another one made by Phil Zimmermann, one by Will Price, one by Jeff Moss, and a number of others by other people. Note that semantically, each of these is equivalent to an X.509 certificate by each of these signers. This entire set of certifications is also present for other email addresses that I have, such as `jcallas@pgp.com` and `jon.callas@pgp.com`.

The self-signed certification allows us to make some interesting statements. This is, for example, where we store the preferences about a key. In these preferences, my key says that when you encrypt to it, I'd like you to use AES-128. If you don't like AES-128, try AES-256. If not that, how about Twofish? These statements allow the owner of a key to tell people who will use it what it would like. At this writing, there is no equivalent of this in X.509 certificates, but the standards people are working on it.

But wait, there's more. An OpenPGP certificate can contain more than one public key. The main key in the certificate has to be some sort of signing key (either a DSA key or an RSA key). There can be other keys for either encryption or signing. You can only use a DSA key for signing, so if your main key is a DSA key, you *must* have another key that can encrypt, be it an Elgamal key or an RSA key. It is possible to use an RSA key for both encryption and signing, but cryptographers don't consider it good form. So we make separate keys for signing and for encryption. It is also possible to have additional keys for signing, as well. The main signing key also makes signatures that state it recognizes these subordinate keys.

The advantage of this approach is that the software can present a unified collection of data that supports several purposes, several names, and several keys while not requiring the user to know all of these details.

Putting it All Together—Constructing Ciphertext from Plaintext

Just as with certificates there are two basic syntaxes for securing a message, the OpenPGP format and the CMS format that is the core of S/MIME [OpenPGP] [CMS]. Despite many differences in the actual format, the pattern of creating a encrypted or signed object is the same. In fact, the basic recipe that I describe here is how almost all encryption and signing works. New systems such as PGP NetShare use this same basic recipe, too. Even very low-level systems such as PGP Virtual Disk and PGP Whole Disk Encryption follow the same basic recipe. Even network systems such as SSL and VPNs use the recipe that follows. Of course, some of the steps might be omitted for some operations. It is possible to encrypt data, sign data, or both. More emails are signed than encrypted, but more files are encrypted than signed. However, nearly every cryptographic system you encounter will use some twist on this basic recipe for encrypting.

1. Start with the plaintext data.
2. Do any data massaging that we need to do. For example, in email, paragraphs have to be appropriately wrapped, the ends of lines have to be turned into the standard Internet form, and so on. In raw, binary data we don't do anything.
3. Create data integrity objects for the massaged data. This means to compute a MAC, a digital signature, or something else equivalent. In general, there is no need for us to do both a symmetric integrity object like a MAC and a digital signature, but it's possible to have them both.
4. Compress the data if desired. OpenPGP compresses the data by default. CMS does not.
5. Encrypt the previous things with a symmetric cipher such as AES. For example, we might encrypt the compressed data with a signature appended to it.
6. Find the public keys that we want to decrypt the data.
7. Encrypt the symmetric key to each of the public keys.
8. Format the raw object. This is where a CMS object becomes an S/MIME mail message, an OpenPGP message gets its email formatting, and so on.

The result of this recipe is something that resembles a set of Russian dolls, or perhaps the layers of an onion. Later steps encapsulate earlier steps.

Taking It All Apart—Getting Plaintext from Ciphertext

When we receive one of these messages, we have to take it apart and come up with the actual plaintext. Here is how we do that.

1. Start with the enveloped ciphertext.
2. Remove any enveloping around the raw, binary ciphertext that was put there by email, web services, and other transports.

3. Scan this list of public keys in the message and look for one for which we have the private key.
4. Use the appropriate public key algorithm to decrypt the message key.
5. Use the message key and the appropriate symmetric-key cipher to decrypt the encrypted message.
6. If the message was compressed, expand it back out again.
7. If there are signatures or MACs, verify them.
8. Look proudly at the plaintext.

Depending on the actual process that we are using, there are variations of this flow. In an encrypted disk, we unwrap the symmetric key when the disk is mounted and use that symmetric key for every disk block we read and write. In email systems, we go through the full recipe with lots of data massaging before and after encrypting and that has to be undone when we decrypt. Even worse, once we are done, we may end up with a complex thing like an email with fonts, pictures, and URLs. However, this is how we take all of the components of cryptography and put them together into a *cryptosystem*.

Going on from Here

We have looked at how we put together a complex cryptographic system from its components and how plaintext gets made into unreadable ciphertext and how that gets turned back into usable plaintext. Of course, there are many details that we haven't looked at and there are other processes we have ignored. Yet many of those are merely another variant on the same framework that we have seen here.

For example, we haven't looked at how your private key gets encrypted to your passphrase. It's an advanced topic. However, we have looked at all the components needed to do this and can think about it. Take the passphrase, use a hash function to get a symmetric key and then encrypt the private key. Simple enough. Of course, there are even more details than that, but if this sort of thing fascinates you, go look at the specifications. I have already provided links here to how OpenPGP (for example) does it, and a link so you can open that up in your web browser. Look for "string to key" in RFC 2440. You know the basics now.

Chapter 5

The Future of Cryptography

Prediction is very difficult, especially about the future.
– Niels Bohr

We've seen where cryptography came from, but where is it going? Of course, it's hard to say, but we can nonetheless talk about the trends. We can also talk about what could perturb the current directions. This gives us both an indication of where we're going, what could make our predictions wrong, and also how likely that is.

A number of the things I'll discuss in this chapter are unsolved problems: challenges that face us with which we have to deal. Some of them are elephants in the room that we're all ignoring, or if not ignoring, noting that they're in the room, take up a lot of space, and do make it hard to dust, but then we go back to the previous subject. Others are genuinely hard problems without a good solution. Still others are tradeoffs. Consider this chapter a tour of stopoffs at interesting problems and surprising things.

From Noun to Adjective, From Syntax to Semantics

Momentum is carrying us so that there will be more cryptography, and more cryptography will be embedded in more places. The larger trend is that there is a change in the metaphor that surrounds the way we use cryptography. It is becoming less of a noun and more of an adjective; it is less of a thing, and more of a modifier. This is the natural progression of all technologies. Ironically, this shift also accompanies greater technological sophistication.

For example, look at radio. When radio was new, a century ago, it was a fascinating thing, it was *The Wireless*, a proper noun used with the definite article. Now, even as it is pervasive, it's a lower-case adjective, not an upper-case noun. I have a wireless telephone in my pocket that can also talk to my computer over a wireless connection that does little more than replace a short wire — Bluetooth. My computer is most often connected to the Internet[1] most often through a wireless network connection. That something is wireless is an important quality but not something that is

[1]The Internet is our age's upper-case noun that will follow this same progression. Already, it is merely controversial to talk about internet things rather than Internet things.

in and of itself interesting. And yet the actual *technology* of wireless, the mechanisms for having pervasive, useful radio for idiotically trivial things like replacing a keyboard cable, is more complex than it has ever been before. About the only thing that we have that is wired any more is power. When we have wireless power for our laptops, the migration will be complete. With a little luck, fuel cells replacing batteries may be precisely the removal of that last wire.

In cryptography, a related change is that we are caring less about syntax and more about semantics. Cryptography is a technology that is itself about language and how it is used, and the constructs we make are becoming more subtle. Cryptography in and of itself is less interesting; it is what we do with the cryptography that is interesting.

Social Expectations

The primary driver of the future direction is the way we expect information to be treated as civilization becomes more dependent on it. We expect the people we buy things from to treat our information with care. We expect health care do the same, but with a more complex task because health information must be instantly available when it is needed, but protected when it isn't. Organizations of all sorts are being required to take care of the information they have.

Europe is the most advanced in terms of legal and social protection of information. The present and upcoming US breach notification laws may very well have a more profound effect on information protection. They do not mandate operational procedures, merely transparency in operational slip-ups. All these requirements are based on a central idea, that information that is encrypted is information that is protected. As this happens, encryption becomes the vehicle that protects all the parties as well as the information itself.

The future of cryptography is entwined with the way we use it, and the way we use it relies on law, custom, regulation, and expectation. We don't have these things very well defined right now. The formalisms, the laws and regulations, are relatively new and they'll get refined in the courts and through custom. That's inevitable, but what we have now is at best a good first draft. There are a lot of rough edges waiting to be filed down. The rough edges are really unsolved problems, so let's look at a number of them.

Digital Signatures and Semantics

Many of the rough edges in the way we'll use cryptography have to do with digital signatures. We're going to be using digital signatures for a lot of things in the future, not for just replacing a written signature but also for many types of data protection.

Digital Signatures Aren't Signatures

Digital signatures are flexible tools that we use for protecting information. The only major problem with digital signatures comes from the name. They aren't signatures. A signature is an *act*, not a thing. When you sign a credit card receipt, for example, it is made legal by the fact that *you* signed

it. What you signed it with is secondary: it could be the legendary X, Zorro's Z, or just about anything else. I have a peeve about those signature scanners used in some cash registers. I write the word "Signature" on them, and do it as fast as I can, so that it doesn't register very well and is only a series of line segments. It also pleases me in an odd way that this is my signature and is unique. Oh, yes, and that I feel like I'm getting away with something, because no one ever calls me on it. No one even cares that it's not the same signature that's on the back of my card. Interestingly, it seems that signatures are rarely compared. Many people I know, particularly those in security have a story about signatures not being checked even when the back of the card says, "always check signature." You can find a delightful discussion of this issue at [SigPrank], in which John Hargrave came up with more and more outrageous challenges to presumptions about signatures.

However, this is less a prank than an observation about the way the world *is* as opposed to the way we perceive it to be. A signature is an act, not a thing. Comparing signatures is merely a secondary security check and it is the *comparing* that is important[2].

A digital signature, in contrast is not an act, it is a thing. It is a thing that we presume can only be produced by one person, but it is really a mathematical, cryptographic calculation. We presume that the "owner" of the private key is the only one who "knows" the key, but let's face it: human beings can't know keys. They can be in possession of some piece of technology that has the only copy of the key, but they don't *know* the key [Seven]. Furthermore, human beings can't do the math that it takes to create the thing. Instead, we have machines to do it for us, which means we need to have a certain amount of faith that the software did the right thing. It's easy for us to examine a piece of paper before we put ink to it; the same isn't true when the document is made of bits.

Most of us know in our hearts that a signature is an act, and I'll demonstrate it to you. At PGP Corporation, one of the things that we have is a picture of my signature. Actually, it's a JPEG image of my ink signature. It's there so that when I'm traveling (as I often am) and some official letter needs to be sent, PGP Corporation can write it up and print it out with my signature on it so that it really came from me. There's nothing *wrong* with this practice, but doesn't it give you the creeps? It does me. Still, when I'm dashing between airplanes, sending the text of a letter to Congress via BlackBerry (which *is* signed with PGP software), how else are they going to put it all together on letterhead and get it out with the last express pickup? It's creepy because a signature is an act, not a thing.

Metaphorically, a digital signature is far more like a wax seal of old than a signature. When we worry about the security of digital signatures, our concerns are like the concerns around a seal. Can a similar one be produced by some bogus signet? Can the seal itself be pried up and moved to some other document? If we could go back in time, it would be much better to call a digital signature a digital seal, but we can't. Therefore we must note that they're not the same[3]. [DSigNotSig]

[2] I believe that the reason why signatures are rarely checked is a simple risk-reward calculation. The clerk has to make a decision about how likely it is that the charge is fraudulent. A false positive here (challenging legitimate customers) is going to insult and may lead to them shopping somewhere else. A false negative — well, there are bad charges all the time, and the costs of such are not borne by the clerk. There are also many more customers than crooks. So with no other evidence, the clerk is better off accepting the charge. And contrary to what you'd think, if someone signs the slip "Mickey Mouse" it is *far* more likely that they're yanking the clerk's chain than that they're a crook.

[3] This confusion isn't new, though. A signet, the thing you use to make a seal in wax comes from the same roots that signing does, and that's obvious enough that I almost thought it went without saying.

The Myth of Non-Repudiation

Whenever you read about digital signatures, it almost always says that digital signatures afford the property of *non-repudiation*, that the signer cannot say they didn't sign it it. This is at best a myth and at worst a mathematician's fantasy. It's not a bad fantasy, however, as fantasies go. Leibniz optimistically espoused the goal of formalizing logic to the point that we could decide disputes by stating things and saying, "Let us calculate." Anyone who has had to deal with a contract dispute can empathize. But just as Gödel dashed Leibniz's plans, let's look at this claim seriously. What *is* non-repudiation?

Take as an example that Bob has a digitally signed document, one signed with Alice's key. Alice bursts into Bob's office and says, "Look, Bob, I know that that letter is signed with my key. I don't deny it. But I didn't sign that letter. I don't know how it came to be, but I didn't do it."

Is Alice telling the truth or not? Did she sign the document? Cryptographically, non-repudiation is the belief that very, very likely, she is not telling the truth. If we assume that she has a 1024-bit RSA key and used a 160-bit hash, then the probability that she is telling the truth is about 2^{-80}. Security-wise, however, there's more to it than that because non-repudiation rests on a number of assumptions. We assume that only Alice has her private key. We assume that Alice's software doesn't have bugs, spyware, etc. We assume that the passphrase on her key has at least 80 bits of strength. These are just the starters, too. We are also assuming that when Alice signed something she fully understood what it was.

In reality, the chance that she's been hacked, or she clicked the wrong button, or her daughter did it, or her secretary did it, or she thought she was signing something else are pretty high. They're a lot higher than 2^{-80}. There's little doubt that software using Alice's key made that signature. Mathematically, that is the reasonable explanation. However, it isn't so easy to know that the software was under Alice's control, or that it was used with Alice's knowledge, understanding, and approval. That takes wisdom and judgment. We have to look at the whole of the situation. If the signed message is one promising to buy the Brooklyn Bridge, it's easier to agree with repudiation than a $10 check for lunch.

Consequently, non-repudiation is merely a mathematical ideal. If someone repudiates a signature, we have to look at the whole context, just like we would with any human commitment. The idea that math gives a property that spares humans from using judgment is a myth.

The Paradox of Stronger Keys

There is one more interesting thing I think about when I consider non-repudiation. If Alice's signature has 80 bits of strength, we consider alternate possibilities such as her system having spyware or the culprit being children or other household gremlins. We're in effect asking the question, "What's more likely: Alice is lying or that she was hacked?" But what about when her signature has 256 bits of strength? Don't we, logically, have to consider alternate scenarios that are less likely than 2^{-80} but more likely than 2^{-256}? Heck, people have had bad reactions to sleeping pills and cooked meals and then gone shopping[4]. And what *are* the odds that space aliens might be

[4]In this particular case, where Alice's mind was asleep but her body was shopping, Alice is almost certainly responsible for the signature no matter how clever the counter-claim. But it brings up an interesting nit in contract law.

playing a prank on her? If the odds of that are 2^{-240}, don't we have to consider that a an alternate explanation as well?

Yes, I'm being puckish with the space aliens, but I think the broader point holds. Stronger keys do not necessarily make the system more secure if we're worried about externalities. If I've entered into the cryptographic Twilight Zone because Alice is happy to pay me for lunch, but insists she didn't make that signature, maybe it's reasonable to wonder if she was befriended by Coyote [CoyoteBlue]. Yes, I'll scan her system for spyware and wonder about her daughter's sense of humor, but when you've removed the likely hypotheses, the unlikely ones are still there – and 2^{-256} is a very small number. Strong cryptography in a chaotic system (like any system involving those darned humans) doesn't necessarily make the system less chaotic, and might actually make it more chaotic. You can't get rid of externalities with encryption, no matter how much encryption you use.

Signatures and Liability

One of the ways digital signatures are being used is to push liability to the signer. This is a reasonable thing to do if you believe in non-repudiation. It's also understandable — who *doesn't* want to push liability and risk to someone else? If you're a bank and your customers use credit cards, you hold the risk. If your customers use digital signatures, they hold the risk.

Of course, the flip side is that as a customer, I like it when banks and merchants assume liability for me. As a customer who is also a cryptographer, I find it irksome that a low-security system like credit cards protect me with insurance, but a high-security system like digital signatures leave me hanging. It doesn't matter how secure the signatures are, the risk is all mine, especially risk from externalities ranging from spyware to bridge-loving space aliens. Stewart Baker called this the "Grandma picks a bad password, Grandma loses her house" problem.

With a bad risk-handling and liability system, only a mad person would go near a public key pair. This situation is bad for us all. Embedded into the monetary system, digital signatures could improve the overall system and reduce losses from fraud and error. But we live in a world in which operating systems have security flaws and criminals are actively looking for credentials to abuse and steal. One of the futures of cryptography is to rethink the laws and customs about digital signatures so that Alice and the Hatter both will want to use them.

A Real-Wold Semantic Shift

We can find a real-world example in semantic subtleties with digital signatures with the new anti-spam mail signature system, DomainKeys Identified Mail (DKIM). Miles Libbey of Yahoo! nicely described the goal of DKIM to be, "we, Yahoo!, want to know that a message for one of our customers claiming to come from eBay really did come from eBay, even if it bounced off of their alumni association." (Full disclosure: I am one of the DKIM specification's authors.)

A DKIM signer is making a statement that takes responsibility for placing the email into the mail stream. A DKIM verifier can use that signature to mean whatever they want it to mean. It is a valuable piece of information that can help process a message. In the case of many senders, that approach will speed the message's transit into the inbox. In others, it may speed the message's transit into the junk mail folder.

DKIM signatures are primarily a conversation between the sending administrative domain and the receiving administrative domain. This is primarily a server-to-server conversation. Although it's possible for the client's machine to verify or even sign a message, this isn't the primary goal. The scope of the signer's responsibility is also limited; the signer says nothing about the content of the message. It is a sort of postmark, a server's statement that says, "I placed this message in the mail stream."

Note that this is a shift in what an email signature means. DKIM is by no means the first attempt to stop email abuse with digital signatures. Most previous discussions have centered on having the end users themselves sign messages, particularly with OpenPGP or S/MIME. In fact, before I got involved with DKIM, I wrote a PGP CTO Corner article about using signatures to stop spam [JCMailSig], arguing that this was not a good idea because of the difficulty of operational management, as well as the semantic dangers of blurring authentication with commitment. If I sign all my messages to show they're not spam, how do I differentiate between idle thoughts (or drafts) and something I am willing to put my name to?

DKIM changes the semantic nature of email signatures in a number of interesting ways:

- A DKIM signature separates the relationship between the message content and the transport envelope. It also separates the end user from the receiving domain. The administrative domain is a broker for the end users, with accountability for those messages going first to the domain. The domain would then take up the dispute with its end users.

- The statements are domain-to-domain. Yahoo! at the very least plans on extending this to the end users (so you can see that the eBay message had an eBay signature), but this isn't necessary.

- DKIM signatures denote a limited commitment, that of having placed the message in the mail stream. After that, many things can happen. For example, let's take Miles's case of the eBay message that goes first to my alumni association. If the server at my university glitches and sends me twenty copies of the eBay message, this is beyond eBay's commitment. Indeed, eBay probably won't even find out about this glitch.

- DKIM signatures are not archival signatures. Although it is possible for an email client to verify a DKIM signature, typically you won't be able to do that years after the fact. The signing keys are stored in DNS[5] and may be removed at any time by the signing domain. We designers consider this to be a privacy-friendly feature, and not a limitation. We were concerned that email authentication could effectively cause all emails to be on-the-record communications, and we didn't want that to happen. The DKIM architecture is such that the signing keys will be closer to ephemeral than epochal. We also consider the domain-to-domain aspects of DKIM to be privacy-friendly features.

- DKIM is based on keys, not certificates. It has a streamlined, abbreviated trust model. A verifier gets the signing key from DNS and if it is in that domain's DNS, then it is valid. DKIM keys aren't revoked; they're merely taken out of their DNS repository.

[5]DNS is the Domain Name System. It is the directory that converts host names and domain names such as www.pgp.com into numeric network addresses.

All of these facets of DKIM are intentional features that are different from other signature systems. DKIM makes statements that are limited in time, limited in scope, and limited in commitment, while still achieving a useful goal: allowing an email receiver to authenticate the message's source.

Cryptography and Reliability

Security systems are one type of system that provides safety. The threats they address are different than other threats. Many threats are unintelligent threats. Lightning, wind, and earthquakes are unintelligent threats. Hackers, spies, and thieves are all intelligent threats. Somewhere in the middle you have semi-intelligent threats that range from insects to vermin to predators. The way we deal with a threat depends on how intelligent it is. Lightning rods solve the threat of fires from lightning and lightning does not develop countermeasures to the lighting rod. Hackers and spies *do* develop countermeasures to solutions we develop and as we design ways to thwart them, we have to take into account that they won't let us put up a lightning rod and be done with it.

Let's differentiate between two broad categories of safety systems: *security* systems that protect against intelligent attackers and *reliability* systems that protect against unintelligent attacks.

We cryptographers have mastered the art of protecting against intelligent attacks. Software such as PGP software is so secure against intelligent attacks that its users have to be careful about the reliability of the overall system: if you forget your passphrase or lose your private key, you have lost the information. One of the challenges cryptographic systems face is to ensure that the protection against intelligent attacks doesn't decrease the reliability of the overall system.

The security of a cryptographic system depends on making sure the wrong people don't have the keys. The reliability of a cryptographic system depends on making sure the right people *do* have the keys. An important future of cryptography is building *key management* systems that look at this critical question, how to make sure the right people always have the keys and the wrong people never have the keys.

The Rise of Hardware

Hardware-enhanced cryptography is coming for a number reasons: the need for security is increasing, hardware designers recognize those needs, and hardware is constantly becoming cheaper and faster. There are also a number of advantages to having cryptography done in hardware. Having hardware do encryption can have an adverse affect on the reliability of the system, particularly if the hardware is also storing keys.

Let me give you an example. Let's suppose you have a laptop with a disk drive that encrypts the data on the disk as it is read and written; it is an on-disk hardware equivalent to our Whole Disk Encryption. Let's also suppose that that disk uses a security chip to store the actual keys that the disk will use and these will be unlocked by the user typing in some passphrase. This process greatly increases the security of the disk. The disk itself does not hold the keys that decrypt the data. The security chip holds encrypted keys, and these are useless without the user's passphrase.

As a security system, this approach has some wonderful characteristics. If you lose the laptop, no one can use your disk (assuming they don't know your passphrase). If you take take the disk out of the laptop and throw it away, there's no chance someone will get to the data on it because they need the keys that are stored in the hardware. The cryptography gives you the equivalent of a shredder for your old disk.

However, this system is less reliable than your old one was. There's the obvious risk that if you forget your passphrase, you have lost your data. This may or may not be an issue. In a large organization, small effects can become large. Let me pull a number out of thin air and say there's a 99.9% chance in a year that a given person will remember his or her passphrase properly. I'm personally willing to accept those odds, but if I am the CIO of a large company, I have to expect some headaches. If there are 100,000 people in my company, then 100 of them will forget their passphrases this year. This is equivalent to 100 disk failures I would not otherwise have had, which is why I would probably require a key management system to have some safe, yet reliable way to manage those systems.

There's also another loss of reliability, and that's in the hardware itself. The security hardware itself could fail, and if it fails, then you lose your disk, too. Additionally, if the hardware is part of your laptop and parts need to be replaced, then your cryptographic secrets are going to be replaced, too. If the video chip, power supply, USB ports, or other components fail on your system, they might take out your disk, too.

Some of the new hardware systems have been recently redesigned with more reliability in mind. Early versions of these hardware systems required that the secrets they held not migrate from chip to chip. If the goal of the system is to bind data to one machine, it works well, but it is bad for a general-purpose security system. The latest version of these systems permit secrets that can be backed up.

Nonetheless, this option creates an operational complexity that didn't exist before. You, the end user, must not only back up your data, but also back up your security hardware. Otherwise, you've done the equivalent of locking your keys in your car, but we clever cryptographers have given you no way to jimmy it open. The future of these systems will be to increase reliability as well as security. Software systems such as PGP software already have reliability components, and the future will bring integration of all these together.

Rights Management

Cryptography is very good at coarse-grained control. If you can't decrypt the data you can't use it, and if you can decrypt it, it's yours. Many people desire a more fine-grained control, which is referred to by the general term *Rights Management*. Most attention is directed to *Digital Rights Management* (DRM), which I can glibly describe as making sure you don't play music through the wrong speakers. Many rights management controversies surround DRM as it applies to entertainment. Therefore, similar systems applied to documents are often called *Enterprise Rights Management* (ERM) systems.

Rights management systems only work against polite attackers. If the content that is being protected is going to be seen or heard, a dedicated attacker can find away to get to it. At the very least, the

attacker can record sound from speakers or photograph a screen. Nonetheless, rights management systems can be effective because the vast majority of possible attackers either don't care enough to go to the trouble of breaking the system, or nonetheless respect the wishes of the protector.

ERM systems create schemes so that documents, mail messages, and other files can only be used under the situations the creator wants. They are attractive because we've all sent mail messages we wish we hadn't, or had a document read by people we didn't want to read it. Everyone wants to be on the protecting end of rights management; no one wants to be on the receiving end.

There are many situations where rights management systems make sense. But there are many social implications of widespread rights management systems, and these will directly affect how much they are used. Here are some examples:

- Rights management encourages and intensifies political behavior. If people can send email that they can revoke, they will. This is one of the selling points of rights management: that it can reduce embarrassment. Yet the potential for embarrassment drives accountability in some people. If they can write something that will only be seen once, they will. The temptation to tell someone off, to imply but not promise support in a meeting, to otherwise weasel an agreement will be too much.

- People will find ways around rights management. Remember: rights management is only effective against polite attackers. If you rile someone with a provocative message, he or she may stop being polite. We live in a world with cameras in telephones. If someone broke an agreement you thought you had with them and recalled the email, you're very likely to take a picture of the next email that person sends you. When the customer service representative tells off a customer (who richly deserved being told off), the customer will take a picture of the mail message and post it on the Web. The presence of rights management will make the offense worse in the public brouhaha.

- Bad actors can game the protected documents. People have fabricated emails to cover their tracks. They can create a picture of an offensive email and defend the fabrication on the grounds that of course there are no tracks of this email existing — the rights management system erases all the tracks.

- Many businesses cannot permit such documents. For example, financial services organizations must keep archives of all communications. They must supply copies of all documents, emails, and even instant message conversations to the regulators when the regulators ask for them. These businesses will have to prevent rights-managed documents from coming into their company.

Most advances in security have increased tracking and accountability. The social implications have involved whether we need that much tracking. Ironically, rights management is a trend that lowers accountability. There are a number of areas where it is useful and desirable. Some people will have to reject rights management because of their own regulatory and legal responsibilities. In other places, rights management will become stigmatized because of dramatic abuses of the system. How much it becomes a norm depends on how well its developers can balance its uses and abuses.

Privacy-Enhancing Technologies

There is another myth that needs to be broken: that there is somehow a tradeoff between security and privacy. The usual argument implies that there is a knob on our society with two poles: "privacy" on one end and "security" on the other—we simply argue over how far the knob gets turned. Certainly, it's possible to increase security while lowering privacy. It's even easier to lower privacy while only giving the appearance of a security improvement—anyone who has been into an airport of late knows what that situation's like.

The assumption one has to exchange privacy for security simply isn't true. There are ways to create and design systems so that they respect privacy as well as security. This subject itself could fill an entire book. But there are also cryptographic systems that are making new ways to create different knobs. Here are a few tastes.

- Perhaps the first significant work on privacy-enhancing security is David Chaum's work on blinded signatures. These digital signatures are made in such a way that they can be verified, but no one, not even the signer, knows which particular signature it is. As an example, it would be useful to have signatures that say, "This person is over twenty-one." I would make an "Over twenty-one" key, and then make these signatures after checking that someone was over twenty-one. These signatures can be verified, but I can't tell myself which particular signature it is. A blinded signature can thus create a credential that is both authenticated and completely anonymous.

  The protocols of how you could use these blinded objects depends on what they are. If the object I signed was a picture, you could compare that picture to the person and the signature itself, and decide that yes, that person is over twenty-one. It could be another security credential, like a PGP key or a certificate, and then a secondary security proof is part of the system.

  No doubt you've figured out the rough edges on this scenario. What happens if that object is loaned to another person? How do we make sure that the object we sign is actually meaningful? How do you get the verifiers to actually accept what we've done?

  Nonetheless, the groundbreaking innovation in blinded signatures is that you can have a digital signature that makes a statement that is verifiable, but unlinkable to other information that is irrelevant. Protecting privacy is all about making sure the receiving party gets only the information they need. [CHAUM]

- Building on the basics of Chaum's work, Stefan Brands created *digital credentials*. [BRANDS] Digital credentials add into the concepts that Chaum created a number of options that protect all parties.

  An objection that many people have to privacy-preserving technologies is the fear the privacy will tempt the holders to misuse the credentials. Consequently, there has to be some way to enforce accountability in the system. Suppose, for example, you've given me an over-twenty-one credential, but if that credential is misused, you can revoke it and fine me — all without knowing precisely which of your customers I am. This possibility puts some real teeth into accountability. It also makes the credential worth more, because all parties know there is enforcement and accountability. Even better, the privacy makes it less likely that you'll make

exceptions for special people. It permits you to make consistent policies and enforce them evenly. Privacy-protection in this case does not lessen security, it *increases* it!

Brands's digital credentials solve these problems and more. They are a major step creating separate knobs for privacy and security.

- The last innovation I'll mention is work by Lea Kissner and Dawn Song on privacy-preserving set operations [KISSNER]. The idea is simple and powerful. Alice and Bob each have a database. They want to know what they have in common without either handing over their database to the other. There are many applications. The US Government has a list of people they don't want to fly[6]. The airlines have a list of passengers. Who is on both lists? A hospital has a list of people on cancer treatment; the state government knows who is on welfare. How does a researcher study the effects of income and disease and comply with laws protecting each group? Or what about two companies that know they share customers and want to help each other with their mutual customers, but don't want to give up their customer lists?

There are many other new forms of privacy-protecting technologies and they are important and exciting because they liberate the way we think about security. They allow us to improve security without the moral erosion that comes with a loss of privacy.

What Will Cause Little Change?

At a certain point in their development, all technologies reach a level where the people who use them see stability in their presentation even though there may be advancement and development in the mechanisms. For example, air travel has changed relatively little since jet airliners became common nearly fifty years ago. It takes five to six hours to fly coast-to-coast across the US You sit in a cramped seat, eat bad food, and try to ignore the people who are trying to control their children. Sometimes your baggage goes walkabout. Despite security changes, lower prices, and declining service, the actual experience of flying is about the same as it's been for the last half-century. Under the covers, of course, the airplanes we fly on are vastly different machines than the early jetliners.

Similarly, as cryptography becomes pervasive, there are going to be many changes and improvements that will cause little outward change. These developments are interesting to cryptographers and engineers, but not to most users. What are some of those developments?

New Hash Functions

Hash functions are flexible functions that cryptographers use for many purposes. In 2004, we learned that the way we're making them isn't good enough. Two years later, we know more about what we don't know about making good hash functions. We are also better at describing how we should be thinking about how to create them. In the next few years, we'll develop new algorithms and deploy them in real-world systems. You, however, won't see much of a change.

[6] Yes, yes, it's a list of names not a list of people and there are many privacy problems with the no-fly list *because* it is merely a list of names. But let's keep that separate.

New Ciphers

Cryptographers are constantly coming up with new ciphers. There are also old systems that we're starting to use more than we did before. As you read about these new developments note that although these changes will spread throughout the use of encryption, to the end user they'll have relatively little effect because the primary change is in the semantics of cryptography, not its syntax.

Encrypt+Authenticate Ciphers

When we construct data, we use separate algorithms to encrypt and to show that the data hasn't been modified. There's work being done on new algorithms that combines these steps so that a single algorithm encrypts the data and can let someone who decrypts it know that the decrypted data is intact. The advantage of these new algorithms is that the data can be processed even as it flows, an approach that brings more efficiencies in network protocols, particularly with streams of live data.

There's similar work going on with public key ciphers on so-called *signcryption* systems that can both sign and encrypt data at the same time.

New and Redesigned Ciphers

Today, we have a good family of ciphers, particularly as a result of the AES competition. Several of the other candidates are completely reasonable ciphers. Of the five finalists, three — Rijndael (the AES proper), Twofish, and Serpent — are even unencumbered by intellectual property issues.

However, there's always advancement in both cryptography and cryptanalysis. Recent work in cryptanalysis is focused not on examining the output of a cipher but on its operation. These observations are called *side-channel analysis*. The results of observing the time it takes for a cipher to run, or how much power the computer running it draws, or the memory used while cipher encrypts are all examples of side-channels.

In the continuous build-and-break game of cryptology, the cryptographers will develop tweaks to ciphers to make them immune to side-channel attacks as well as new ciphers that have immunities to side-channels. As you might expect, the cryptanalysts will also come up with new side channels and new ways to apply side-channels to new situations.

Elliptic Curve Cryptography

A chain is only as strong as its weakest link, the saying goes. This statement is true for cryptography, and the different components we use should be of about the same strength. Today, we commonly use public-key systems that have much larger keys than a symmetric key of similar strength. We're also starting to use symmetric keys that are 256 bits in length. For RSA and DSA, a 3,000-bit public key balances with a 128-bit symmetric key, but it takes a 15,000-bit public key to match a 256-bit symmetric key. 15,000 bits is a very large key. It would be impractical to use in small devices like telephones or door knobs. Because of this stretch, the US government is encouraging

the use of elliptic curve cryptography. With elliptic curve systems, a 512-bit public key balances a 256-bit symmetric key. The shift to elliptic curve systems will happen over the next five to fifteen years, as much because of the nest of patents surrounding such systems.

Bi-Linear Map Ciphers

Just as there is development on symmetric ciphers, there's also development in public-key ciphers. The most interesting is in the use of *bi-linear maps*. They are a different class of elliptic curve than the ones being used in more traditional elliptic-curve cryptography and with different mathematics. They allow creating a family of keys from a master private key. Each of the key family is a public key pair; it has both a private key and a public key. Additionally, the owner of any private sub-key cannot learn other private sub-keys. Thus, they share many of the properties of symmetric keys as well as being public key systems. There are many uses for these ciphers including:

- A user creates a single master public key and many throw-away keys that can be used for a variety of purposes while needing only to store that single key. Despite the fact that storage is cheap and becoming exponentially cheaper, there are still places where this situation is desirable because it can simplify the data management in a hierarchical system. Additionally, the master key owner can give key pairs to other people that that master key owner can use to communicate with those other people.

- A server creates a single master public key and uses the same mechanisms to issue keys to all the users of that system. This setup creates an escrowed key system descended from the master key. Any of the people or subsystems that have subkeys can securely work with each other, but are still under the control and observation of the master key holder.

  These scenarios have privacy issues when the subordinate keys are owned by people, because it builds a cryptosystem with an obvious backdoor. There are many cases when this outcome is not a problem, however. For example, in operating security, subsystems or virtual machines can all have their own keys. Clusters of computers can also have their own unified set of keys while minimizing the number of touch points of the cryptographic subsystems.

- Bi-linear maps can have their subkeys be nearly anything at all. So why not have them be the hash of an arbitrary string? This approach allows us to create an *identity-based encryption* systems where names are effectively keys. Of course, that means we're trusting that the hash function is a good hash function because a collision in the hash of names means a collision of keys. However, none of the hash function weaknesses we are facing today are a threat as yet.

Bi-linear maps are the most interesting new development coming because they both simplify and complicate key management. Typically, they tend to simplify the key issuer's job while complicating the end user's key management. The issuer needs only one key, but the user may easily end up with at least one key per issuer, and often more. The hard part of cryptography has always been key management, so this is no small issue. It is bad for the less-capable users in a system end up with the heavy lifting in key management. Bi-linear map systems also have vastly more complex revocation problems because revoking or retiring a master key can force a change of many keys with no good way to know where they are.

In the cases where the bi-linear map creates an identity-based encryption scheme, this problem can even be worse because naming is the hardest part of key and certificate issuance. It is relatively easy for me to argue that I deserve the "Jon Callas" key, but who gets the "John Doe" key? Furthermore, there are other complex issues. Even if we know who gets the "John Doe" key for 2006, who gets the "John Doe" key for 2106? Unfortunately, the most important thing about key management is the metadata that is associated with the key. A name is only one piece of the metadata, and the most arbitrary and problematic piece. Our lives are complicated with account numbers, handles, user names, and tags because we can't manage by names alone.

The new possibilities bi-linear maps open up provide not only room for advancement, but for discussion as well.

Quantum Cryptography, or Perhaps Quantum Secrecy

Quantum cryptography is an exciting new discipline that despite the name is not actually cryptography. Perhaps this is a pet peeve of mine, but cryptography is the art and science of using mathematics to make messages unreadable. Quantum cryptography should be called quantum secrecy; it is a way to use quantum effects to keep messages from being intercepted.

The techniques use quantum effects to entangle the photons, binding them together. If an observer who is not one of the two main parties tries to observe the photons as they transmit, they lose the information they were carrying. Thus, any photon that completes the path between the two parties has not been observed by an intermediary.

This is, of course, a dramatic oversimplification. Nonetheless, quantum secrecy has the promise of becoming an alternative to cryptography. If I could teleport a letter directly into your hand, there's no need for us to encrypt it as it goes from my hand to yours.

As amazing as this technology is, it still has only a niche market. Modern networking takes place over a variety of media. As I write this book, versions of it are being sent between me and my editor over a network that using radio, electricity through copper, and photons through glass. Quantum secrecy cannot replace all of those; I still need to encrypt the manuscript. While it will be invaluable in many uses, it won't change the general-purpose use of cryptography.

What Could Change the Course?

Part of the reason that Niels Bohr is right that it's so hard to make predictions about the future is that the future is so darned unpredictable. It is therefore slightly absurd to try to predict the unpredictable. Despite that, here are a few wild cards that give me good excuses for being wrong.

The Effect of Patents

Many of the systems I describe above are patented. In fact, the majority of interesting innovations in cryptography have been patented, are patented, or will be patented (as soon as they're invented). On the one hand, patents are good because they're how inventors can actually own their invention. On

the other hand, this ownership often slows down deployment. Many advancements in encryption have languished because of patent issues. Patents can create fortunes, and they can also stop a brilliant system.

Science-Fictional Technology

I do not mean the term "science-fictional" to be at all pejorative. It was not all that long ago, after all, that going to the moon was science fiction and not nostalgia. And in those days, uncrackable ciphers would be science fiction.

Today, people are working on making some forms of science fiction into fact, and these could have a dramatic effect on the way we do cryptography. The most obvious technology in this category is quantum computing. There is active research into quantum computing, but there is still not a lot of detail known about what quantum computers can actually *do*.

Good descriptions of quantum computing can be found at [QC-WEST] and [QC-BESM]. We all agree that quantum computers would be able to do some calculations with amazing amounts of parallelism. Peter Shor has devised algorithms that can do mathematical calculations that are feasible on a quantum computer, but not feasible on today's computers. Among them, he has devised ways to factor numbers and compute discrete logarithms [SHOR]. If you could build such a machine, it could break the public keys we use today.

However, there are also people at work on what is now called *post-quantum-computing cryptography*. There are public key schemes, such as those involving chains of hashes, that would be immune to attacks on quantum computers. It is also possible there are twists to the present systems that would also make them usable post-quantum computing.

Similarly, there are people working on computations with DNA and other biological systems. Interestingly, Leonard Adleman, the "A" in RSA, is a researcher in DNA computing. In theory, DNA computing can also solve some problems that would be difficult to solve on an ordinary computer, like factoring. There is also work on teleporting particles, which could change the way electrical or optical computers work.

Any of these developments could throw any predictions of where we are going into a into the trash. And so could things that we haven't thought of at all. We just don't know what the future will bring.

Legal Changes

Changes in the law reflect something even more unpredictable than quantum states: the way people think about things. A decade ago, expectations about cryptography made it be governed a certain way. The world economy is shifting to being dependent on ideas as much as it is on physical things. Ubiquitous cryptography has been part of that shift. The terrorist attack of 2001 did not change that trend, and there are presently no signs that this will change. Many of our present concerns are about making sure that information is protected and this means cryptography.

Of course, these societal trends can change. But such a change would also come with a change in attitude about free trade, globalization, and a drive toward more economic efficiencies. I don't

think this transformation is likely, but it could happen. If it does, I can only hope that someone looking back on what I write now would add that of course those changes were unpredictable.

Additional Reading

[ADFGVX] J. R. Childs, *General Solution of the ADFGVX Cipher System*, Aegean Park Press, Book C-88, 245pp, ISBN: 0-89412-284-3.

The ADFGVX cipher was a cipher that the Germans developed during WWI that consisted solely of the six letters that give it its name.

[AEGEAN] A source of many good books is the Aegean Park Press, <http://www.aegeanparkpress.com/desc.html>. Their books include text books from the US, British, French, and Italians. Most of these have only been declassified in the last few years. They also include histories and analyses of cipher bureaus in WWI as well as other times, such as a book on the cryptosystems used by each of the candidates in the 1876 US Presidential election. If you're interested enough in the history of cryptography to bother reading this citation, there is something in their catalog that will fascinate you.

[AES] Daemen, Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*, Springer-Verlag, 2002.

[AESCOMP] NIST's archived pages about the AES competition can be found at <http://csrc.nist.gov/CryptoToolkit/aes/>. The list of requirements for the AES are at <http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes_9709.htm>.

[AHS] The best place to start to read about the Advanced Hash Standard competition is <http://www.nist.gov/hash-function>. Expect it to change a lot.

[ALICE] John Gordon, *The Alice and Bob After Dinner Speech, given at the Zurich Seminar, April 1984*,, <http://downlode.org/etext/alicebob.html>.

[AUPRIV] The Privacy Law and Policy Reporter web site maintains a detailed description of Australian laws at <http://austlii.edu.au/~graham/PLPR_australian_guide.html>.

[Birthday] A very nice discussion of the Birthday problem can be found at <http://mathforum.org/dr.math/faq/faq.birthdayprob.html>. Possibly more math than you can stand describing all of the statistics is at <http://mathworld.wolfram.com/BirthdayProblem.html> with a direct <http://mathworld.wolfram.com/BirthdayAttack.html>.

[BRANDS] Stefan Brands, *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*, MIT Press, ISBN 0-262-02491-8.

You can also find a PDF version of the book at the site of Brands's company, *Credentica* <http://www.credentica.com/the_mit_pressbook.php>.

[BPTrust] The Bletchley Park Trust can be found at <http://www.bletchleypark.org.uk/>.

[CANPRIV] Fact Sheet on Canada's Privacy Laws: <http://www.privcom.gc.ca/fs-fi/02_05_d_15_e.asp>.

The Canadian Privacy Resource Centre contains a number of documents about privacy laws in Canada: <http://www.privcom.gc.ca/information/02_06_01_e.asp>.

Canadian Privacy Act itself: <http://laws.justice.gc.ca/en/P-21/index.html>.

[CCTrust] The Codes and Ciphers Heritage Trust can be found at <http://www.bletchleyparkheritage.org.uk/>.

[CHAUM] David Chaum, *Achieving Electronic Privacy*, Scientific American, August 1992, p. 96-101. <http://www.chaum.com/articles/Achieving_Electronic_Privacy.htm>.

David Chaum, *Security Without Identification: Transaction Systems to Make Big Brother Obsolete*, Communications of the ACM, Vol. 28, No. 10, pages 1030-1044; October 1985. <http://www.chaum.com/articles/Security_Wthout_Identification.htm>.

[Codebooks] Jim Reeds, *Commercial Telegraphic Code Books*, <http://www.dtc.umn.edu/~reedsj/codebooks.html>

This is a fun web site that has a lot of good information on the way the code books were used to make telegrams cheaper, as well as to obscure the meaning of sensitive ones.

[CMS] R. Housley, *Cryptographic Message Syntax (CMS)*, RFC3852 <http://www.ietf.org/rfc/rfc3852.txt>.

[Colossus] <http://www.bletchleyparkheritage.org.uk/ColRbd.htm>.

[CoyoteBlue] Christopher Moore, *Coyote Blue*, Perennial Books, ISBN 0-06073-543-0.

All right, there isn't any cryptography in Christopher Moore's book, but if you haven't read his hilarious fiction, you should.

[CRYPTOAG] There is much information and speculation about the Crypto AG scandal. A good place to start is with this article <http://www.aci.net/Kalliste/speccoll.htm>. An article from *Der Spiegel* can be found in German here <http://jya.com/cryptoag.htm> with an English translation here <http://jya.com/cryptoa2.htm>. A *Baltimore Sun* article can be found here <http://jya.com/nsa-sun.htm>. The *Covert Action Quarterly* has a report as well at <http://mediafilter.org/caq/cryptogate/>.

[CryptoPolicy] *A Brief History of Cryptography Policy*, <http://www.nap.edu/readingroom/books/crisis/E.txt>.

[CSIZE] Arjen K. Lenstra, Eric R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology Volume 14, Number 4, pp255-293, 2001.

Also see the web site <http://www.keylength.com/> which has an interactive demonstration that allows you to tweak the parameters and assumptions of what is the best key size.

[DIFCRYPT1] Eli Biham and Adi Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Journal of Cryptology, vol. 4, pp. 3-72, IACR, 1991.

[DIFCRYPT2] Eli Biham and Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 188pp, ISBN 0-38797-930-1.

[DKIM] The DKIM association can be found at <http://www.dkim.org/>.

[DSigNotSig] Jon Callas and Bruce Schneier, *Why Digital Signatures Are Not Signatures*, The Industry Standard, 2000.

A version of this article is on Bruce's site at <http://www.schneier.com/crypto-gram-0011.html#1>.

[ECC] Certicom, who have the most experience in commercializing elliptic curve cryptography have an excellent web site explaining the basics. See <http://www.certicom.com/index.php?action=ecc_tutorial,home> for their tutorial on the math behind elliptic curve cryptography.

[EME] Shai Halevi and Phillip Rogaway, *A Parallelizable Enciphering Mode*, <http://eprint.iacr.org/2003/147>.

[EMovie] *Enigma*, Directed by Michael Apted, Screenplay by Tom Stoppard from the novel by Robert Harris. <http://imdb.com/title/tt0157583/>.

[EnigmaBP] BletchleyPark.net *Enigma*, <http://www.bletchleypark.net/stationx/enigma.html>.

[EnigmaDM] Hartmut Petzold, *The Enigma rotor-type ciphering machine of the German Armed Forces*, Deutsches Museum, <http://www.deutsches-museum.de/ausstell/meister/e_enigma.htm>.

[EUDPD] The full text of the EU Data Privacy Directive is here: <http://www2.echo.lu/legal/en/dataprot/directiv/directiv.html>.

An EU central repository for much information is at: <http://europa.eu.int/comm/justice_home/fsj/privacy/>.

An American perspective on the EU-DPD which has a good summary is here: <http://www.dss.state.ct.us/digital/eupriv.html>.

[GPlant] Peter Gutmann, *The Crypto Gardening Guide and Planting Tips*, February 2003, <http://www.cs.auckland.ac.nz/~pgut001/pubs/crypto_guide.txt>.

Peter Gutmann is a brilliant researcher who does more than just cryptography. He was one of the developers of PGP 2, among many other achievements. He also has an acid wit and is not afraid to use it. All of his writings are well worth reading. This is a good thing to read if you are or are contemplating writing anything with cryptography in it. If you work with protocols, you will be enlightened by at least one of these tips.

[Hagelin] Wayne G. Barker, *Cryptanalysis Of The Hagelin Cryptograph*, Aegean Park Press, Book C-17, 223pp, ISBN: 0-89412-022-0.

[Hellman] Martin Hellman's description of how he got involved in cryptography can be found at <http://www-ee.stanford.edu/~hellman/crypto.html>. Other pages in his web site, including <http://www-ee.stanford.edu/ hellman/breakthrough.html> are important to read to understand The Crypto Wars.

[IDTheft] Privacy Rights Clearing House, *How Many Identity Theft Victims Are There? What **is** the Impact on Victims?*, <http://www.privacyrights.org/ar/idtheftsurveys.htm>.

[JCMailSig] Jon Callas, *Crypto and Spam*, <http://www.pgp.com/library/ctocorner/cryptoandspam.html eg

[JPPRIV] The Privacy Exchange has official and unofficial translations of PIPA as well as surveys on attitudes about privacy at <http://www.privacyexchange.org/japan/japanmain.html>.

[JWheel] The Jefferson Wheel cipher is nicely described at <http://www.monticello.org/reports/interests/wheel_cipher.html>. You can read about how to create one of your own out of paper and paper cups at <http://www3.brinkster.com/Redline/crypt/jefferson.asp>.

[KAHN] D. Kahn, *The Codebreakers: The Story of Secret Writing*, Simon & Schuster Trade, 1996, ISBN 0-684-83130-9 (updated from the 1967 edition).

[KERCKHOFFS] Auguste Kerckhoffs, *La Cryptographie Militaire*, Journal Des sciences Militaires, Janvier 1883, <http://www.petitcolas.net/fabien/kerckhoffs/>.

Alors, this article is in French, but it contains the first statement of Kerchoffs's Principle, the idea that only keys should be secret in a cryptographic system. That core idea, that security systems should be "white boxes" rather than "black boxes" is core to the advances we have had in security over the last few decades.

[KISSNER] Lea Kissner and Dawn Song, *Privacy-Preserving Set Operations*, CMU-CS-05-113, June 2005. <http://www.cs.cmu.edu/~leak/papers/set-tech-full.pdf>.

I wrote an article for the PGP CTO Corner about Kissner and Song's work, which you can find at <http://www.pgp.com/library/ctocorner/sets.html>.

[KSHASH04] John Kelsey and Bruce Schneier, *Second Preimages on n-bit Hash Functions for Much Less than 2^n Work*, <http://eprint.iacr.org/2004/304 eg

[LEVY] Steven Levy, *Crypto: How the Code Rebels Beat the Government–Saving Privacy in the Digital Age*, Diane Pub Co, 356pp, ISBN: 0-75675-774-6.

[MAGIC] Frank B. Rowlett, *The Story Of Magic, Memoirs of an American Cryptologic Pioneer*, Aegean Park Press, Book C-81, 266pp, ISBN: 0-89412-273-8.

This book is the story of Frank B. Rowlett, who lead the team that broke the Japanese PURPLE diplomatic cipher. It is a particularly good book, as Rowlett describes what it was like to work on it in great detail. Also, unlike the much better-known Enigma cryptanalysis, no PURPLE machine was ever found, as the Japanese were quite skilled in destroying them before they were captured. The only PURPLE machines in existence are Rowlett's recreations of them from ciphertext alone!

[MARKS] Leo Marks, *Between Silk and Cyanide: A Codemaker's War, 1941-1945*, Free Press, 624pp, ISBN: 0-684-86422-3 (hard), 0-684-86780-X (paper)

It is hard to say enough good things about this book. Leo Marks was a remarkable man. His parents ran the famous London bookshop at 84 Charing Cross Road. He was a screen writer, occasional actor (including the voice of Satan in *The Last Temptation of Christ*), and he ran the codes security for SOE in WWII, at the age of twenty-two.

I ran across this book shortly after it was published in the US, just as the dot-com boom was busting. I almost didn't read it because it is one of my peeves that we in security pay too much attention to the military and banks and not enough to the real world. From its opening paragraph, it is wryly funny, acidly angry. The story is good, and oh,

yes, somewhere in there Marks taught me to rethink the way I was thinking about security.

Marks taught the brave, tragic people being dropped behind enemy lines about information security, which their very lives depended on, and they didn't listen. He fought stupid bureaucracy over procedures everyone knew were wrong. The title comes from Marks's use of one-time pads printed on silk (easy to hide, easy to burn) for usable, secure ciphering. His charges' only choice was between silk and cyanide. If you're in the security biz, you probably aren't sending people off inadequately prepared for near-certain death, they're probably only inadequately prepared for near-certain business screwups. I learned that people will not make good security decisions on their own, even — *especially* — if their lives depend on it.

It feels stupid and trivial to say so, but much of the design of PGP systems since then comes from lessons Leo Marks teaches so well. This is not only a good read, but it made me a better security professional for having internalized what happened to Marks.

[MAURER] Ueli Maurer, *Modeling a Public-Key Infrastructure*, Proceedings of the 1996 European Symposium on Research in Computer Security (ESORICS' 96), Lecture Notes in Computer Science, Springer-Verlag, vol. 1146, pp. 325-350, Sep 1996. <http://citeseer.ist.psu.edu/maurer96modelling.html eg

[NSENC] Bruce Schneier, *Non-Secret Encryption*, Crypto-Gram of May 1998, <http://www.schneier.com/crypto-gram-9805.html>.

[OpenPGP] Jon Callas, Lutz Donnerhacke, Hal Finney, and Rodney Thayer, *OpenPGP Message Format*, RFC2440 <http://www.ietf.org/rfc/rfc2440.txt>.

This is the standard for the core OpenPGP data protocols. It describes how actual OpenPGP messages, as created by PGP and other systems, are put together and taken apart.

[OPGPMIME] M. Elkins, D. Del Torto, R. Levien, T. Roessler, *MIME Security with OpenPGP*, <http://www.ietf.org/rfc/rfc3156.txt>

This is the standard for OpenPGP/MIME, the way that OpenPGP is formed into complex email messages. It builds upon OpenPGP Formats.

[ORPGP] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995, 393pp, ISBN 1-56592-098-8.

[PGP2] P. R. Zimmermann, *The Official PGP User's Guide*, The MIT Press, 1995, 216pp, ISBN 0-262-74017-6.

This is Phil Zimmermann's original PGP book. Sadly, it is presently out of print, but copies can be found from used book stores.

[PGP2S] P. R. Zimmermann, *PGP: Source Code and Internals*, The MIT Press, 1997, 933pp, ISBN 0-262-24039-4.

This is the source code book for PGP 2.6. It is also out of print.

[PGPsource] The sources for PGP can be downloaded from <http://www.pgp.com/downloads/sourcecode/>. This includes not only the sources to PGP Desktop, but the PGP Command Line sources and the PGP Universal GPL-modified sources. You can also find PGP's policies on assurance and special build requirements at <http://www.pgp.com/company/pgpassurance.html>.

[QC-BESM] A. Barenco, A. Ekert, A. Sanpera and C.Machiavello, *A Short Introduction to Quantum Computation*, originally in *La Recherche*, November 1996. Adapted by A. Barenco and available at <http://www.qubit.org/library/intros/comp/comp.html >.

[QC-WEST] Jacob West, *The Quantum Computer, An Introduction*, <http://www.cs.caltech.edu/~westside/quantum-intro.html>.

[QUBIT] The Centre for Quantum Computing at Oxford University has many resources, as well as links to Cambridge, Caltech and other places. There are many good articles here. <http://www.qubit.org/ >.

[RABIN] Neal R. Wagner, *The Laws of Cryptography: Rabin's Version of RSA*, <http://www.cs.utsa.edu/~wagner/laws/Rabin.html> Wagner declares the site "obsolete," but it is still a useful read. A draft of his full book can be found at <http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf>.

[SCHNEIER] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, second edition, John Wiley & Sons, 1996; ISBN 0471117099.

[Seven] George A. Miller, *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*, originally published in The Psychological Review, 1956, vol. 63, pp. 81-97, reproduced at <http://www.well.com/~smalin/miller.html>, with the author's permission, by Stephen Malinowski.

[SHOR] Peter W. Shor, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings of the 35th Symposium on the Foundations of Computer Science (1994), 124-134. <http://citeseer.ist.psu.edu/14533.html>.

[SigPrank] John Hargrave, *The Credit Card Prank*, <http://www.zug.com/pranks/credit/>.

[SINGH] S. Singh, *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*, Doubleday & Company, Inc., 1999, ISBN 0-385-49531-5.

[Slater] Rick Fowler, *Slater's Telegraphic Code*, <http://homepage.usask.ca/~rhf330/tele.html>. Slater's code also incorporated a simple cipher in it as well, and was used during the Riel Rebellion or Northwest Resistance in Canada in the 1880s.

[STEGO] Neil F. Johnson and Sushil Jajodia, *Steganography: Seeing the Unseen*, IEEE Computer, February 1998, pp26-34, <http://www.jjtc.com/pub/r2026a.htm>. See also Neil Johnson's web site on steganography at <http://www.jjtc.com/stegdoc/>.

[SY05] Michael Szydlo and Yiqun Lisa Yin, *Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing*, <http://eprint.iacr.org/2005/248>.

[TAR] Unfortunately, there is no definitive description of the `tar` file format despite its near-ubiquitous use and the fact that it is part of the POSIX 1003.1-1990 standard. PGP Zip's `tar` implementation is reverse engineered from *gnutar*, <http://www.gnu.org/software/tar/>, which is not entirely the same as POSIX `tar`. Within the *gnutar* package, the file `dist/src/tar.h` has a description of `tar` structures, but unfortunately, the code is the best documentation.

[TEDIUM] Just in case you have no patience for the one-time pad example, the message encrypted is 'NOW IS THE TIME FOR ALL GOOD CRYPTOGRAPHERS TO COME TO'.

[TLS] T. Dierks and C. Allen, *The TLS Protocol Version 1.0*, RFC2246 <http://www.ietf.org/rfc/rfc2246.txt>.

[TWOFISH] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson, *Twofish: A New Block Cipher*, <http://www.schneier.com/twofish.html>.

Note that they describe it as a 128-bit algorithm, but with 192- and 256-bit modes. See — 128 bits is enough.

[Unicode] Unicode is a computer code that has as its goal to have a number for every symbol that is used in all human languages. You can find more from the Unicode Consortium's web site at <http://www.unicode.org/>.

[URBAN] Mark Urban, *The Man Who Broke Napoleon's Codes*, HarperCollins, 368pp, ISBN: 006018891X. This is the story of George Scovell, an officer in Wellington's army who broke the codes and ciphers that were used by the French in Iberia. This is an interesting tale because all of the techniques described to improve human cryptography were used one way or another by the French, and Scovell gradually beat them all. This is also interesting because you can see how some improvements when carelessly implemented can actually make the cryptanalyst's job easier.

[VENONA] The US Army started in 1943 to attack Soviet diplomatic messages in a project they code-named VENONA. It completed in 1980 and was declassified in 1995. The NSA site for the project is at: <http://www.nsa.gov/venona/index.cfm>.

[WANG04] Xiaoyun Wang and Dengguo Feng and Xuejia Lai and Hongbo Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*,<http://eprint.iacr.org/2004/199>.

[WANG05] Xiaoyun Wang, Hongbo Yu, Yiqun Lisa Yin, *Finding Collisions in the Full SHA-1*, Advances in Cryptology — CRYPTO 2005, LNCS 3621, Springer, 2005, ISBN 3- 540-28114-2, pp17-36.

[WWPG] The Privacy Law and Policy Reporter web site has a survey of worldwide and international privacy issues. It can be found at: <http://austlii.edu.au/~graham/PLPR_world_wide_guide.html>.

[ZUSE] Prof. Horst Zuse, *The Life and Work of Konrad Zuse*, <http://www.epemag.com/zuse/>.