404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 1

Chapter 1

Introduction

Solutions in this chapter:

- Threat Models
- What Is Cryptography?
- Asset Management
- Common Wisdom
- Developer Tools

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 2

Introduction

Computer security is an important field of study for most day-to-day transactions. It arises when we turn on our cellular phones, check our voice mail and e-mail, use debit or credit cards, order a pay-per view movie, use a transponder through EZ-Pass, sign on to online video games, and even during visits to the doctor. It is also often used to establish virtual private networks (VPNs) and Secure Shell connections (SSH), which allows employees to telecommute and access computers remotely.

The use, and often misuse, of cryptography to solve security problems are driven by one cause: the need for security. Simply *needing* security does not make it so, a lesson all too often learned after the fact, or more importantly, after the exploits.

> Notes from the Underground…
>
> **Known Exploit—*Dark Age of Camelot***
>
> URL: http://capnbry.net/daoc/advisory20040323/daoc-advisory2.html
>
> In March 2004, an exploit for the video game *Dark Age of Camelot* (Mythic Entertainment) made use of the weak server authentication the game used to perform secure billing transactions. It allowed attackers to intercept communication between a real server and client and read all the private billing data.
>
> Even though the developers used a known and tested cryptographic library to provide core algorithms, they had used the algorithms incorrectly. As a result, the attackers did not have to break hard cryptographic algorithms such as RSA or RC4, just the weak construction in which they were used.

The Mythic exploit is a classic example of not knowing how to use tools properly. It is hard to fault the developer team. They are, after all, video game developers, not fulltime cryptographers. They do not have the resources to bring a cryptographer on team, let alone contract out to independent firms to provide the same services.

The circumstances Mythic was in at the time are very common for most software development companies throughout the world. As more and more small businesses are created, the fewer resources they have to pool on security staff. Security is not the goal of the end-user product, but merely a requirement for the product to be useful.

For example, banking hardly requires cryptography to function; you can easily hand someone $10 without first performing an RSA key exchange. Similarly, cell phones do not require cryptography to function. The concept of digitizing speech, compressing it, encoding the bits for transmission over a radio and the reverse process are done all the time without one thought toward cryptography.

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 3

Because security is not a core product value, it is either neglected or relegated to a secondary "desired" goal list. This is rather unfortunate, since cryptography and the deployment of is often a highly manageable task that does not require an advanced degree in cryptography or mathematics to accomplish. In fact, simply knowing *how* to use existing cryptographic toolkits is all a given security task will need.

Threat Models

A threat model explicitly addresses venues of attack adversaries will exploit in their attempts to circumvent your system. If you are a bank, they want credentials; if you are an e-mail service, they want private messages, and so on. Simply put, a threat model goes beyond the normal use of a system to examine what happens on the corner cases. If you expect a response in the set *X*, what happens when they send you a response *Y* that does not belong to that set?

The simplest example of this modeling is the *atoi*() function in C, which is often used in programs without regard to error detection. The program expects an ASCII encoded integer, but what happens when the input is not an integer? While this is hardly a security flaw, it is the exact sort of corner cases attackers will exploit in your system.

A threat model begins at the levels at which anyone, including insiders, can interact with the system. Often, the insiders are treated as special users; with virtually unlimited access to data they are able to commit rather obtuse mistakes such as leaving confidential data of thousands of customers in a laptop inside a car (see, for instance, http://business.timesonline.co.uk/article/0,,13129-2100897,00.html).

The model essentially represents the *Use Cases* of the system in terms of what they potentially allow if broken or circumvented. For example, if the user must first provide a password, attackers will see if the password is handled improperly. They will see if the system prevents users from selecting easily guessable passwords, and so on.

The major contributing factor to development of an accurate threat model is to not think of the use cases in terms of what a proper user will do. For example, if your program submits data to a database, attackers may try an *injection attack* by sending SQL queries inside the submitted data. A normal user probably would never do that. It is nearly impossible to document the entire threat model design process, as the threat model is as complicated if not more so than the system design itself.

This book does not pretend to offer a treatment of secure coding practices sufficient to solve this problem. However, even with that said, there are simple rules of threat model design developers should follow when designing their system:

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 4

Simple Rules of Threat Model Design

1. How many ways can someone transition into this use case?
   i. Think outside of the intended transitions.
   ii. Are invalid contexts handled?
2. What components does the input interact with?
   i. What would "invalid inputs" be?
3. Is this use case effective?
   i. Is it explicitly weak? What are the assumptions?
   ii. Does it accomplish the intended goal?

What Is Cryptography?

Cryptography is the automated (or algorithmic) method in which security goals are accomplished. Typically, when we say "crypto algorithm" we are discussing an algorithm meant to be executed on a computer. These algorithms operate on messages in the form of groups of bits.

More specifically, people often think of cryptography as the study of ciphers; that is, algorithms that conceal the meaning of a message. Privacy, the actual name of this said goal, is all but one of an entire set of problems cryptography is meant to address. It is perhaps most popular, as it is the oldest cryptography related security goal and feeds into our natural desire to have secrets. Secrets in the form of desires, wants, faults, and fears are natural emotions and thoughts all people have. It of course helps that modern Hollywood plays into this with movies such as *Swordfish* and *Mercury Rising*.

Cryptographic Goals

However, there are other natural cryptographic problems to be solved and they can be equally if not more important depending on who is attacking you and what you are trying to secure against attackers. The cryptographic goals covered in this text (in order of appearance) are privacy, integrity, authentication, and nonrepudiation.

Privacy

Privacy is the property of concealing the meaning or intent of a message. In particular, it is to conceal it from undesired parties to an information transmission medium such as the Internet, wireless network link, cellular phone network, and the like.

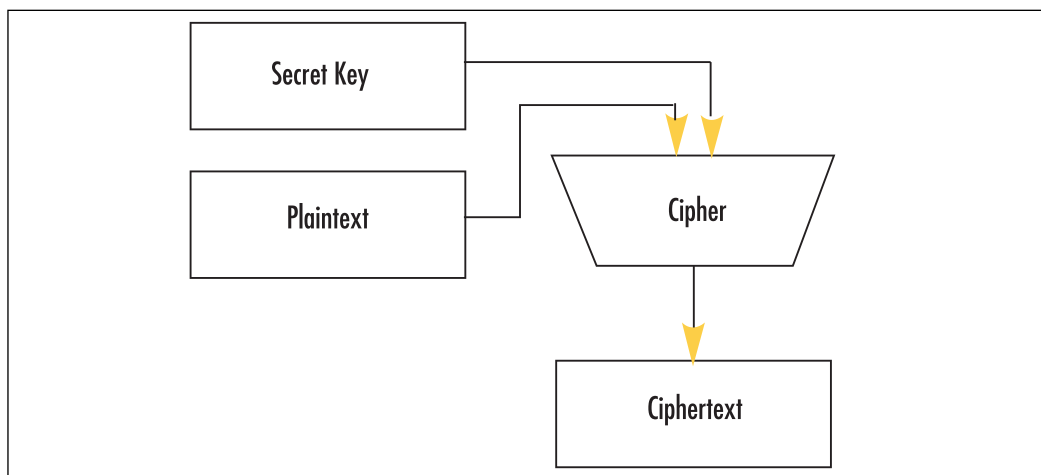
Privacy is typically achieved using *symmetric key ciphers*. These algorithms accept a secret key and then proceed to encrypt the original message, known as *plaintext*, and turn it into piece of information known as *ciphertext*. From a standpoint of information theory, cipher-

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 5

text contains the same amount of entropy (uncertainty, or simply put, information) as the plaintext. This means that a receiver (or recipient) merely requires the same secret key and ciphertext to reconstruct the original plaintext.

Ciphers are instantiated in one of two forms, both having their own strengths and weaknesses. This book only covers *block ciphers* in depth, and in particular, the National Institute for Standards and Technologies (NIST) Advanced Encryption Standard (AES) block cipher. The AES cipher is particularly popular, as it is reasonably efficient in large and small processors and in hardware implementations using low-cost design techniques. Block ciphers are also more popular than their stream cipher cousins, as they are universal. As we will see, AES can be used to create various privacy algorithms (including one mode that resembles a stream cipher) and integrity and authentication algorithms. AES is free, from an intellectual property (IP) point of view, well documented and based on sound cryptographic theory (Figure 1.1).

Figure 1.1 Block Diagram of a Block Cipher



NOTE

URL: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/

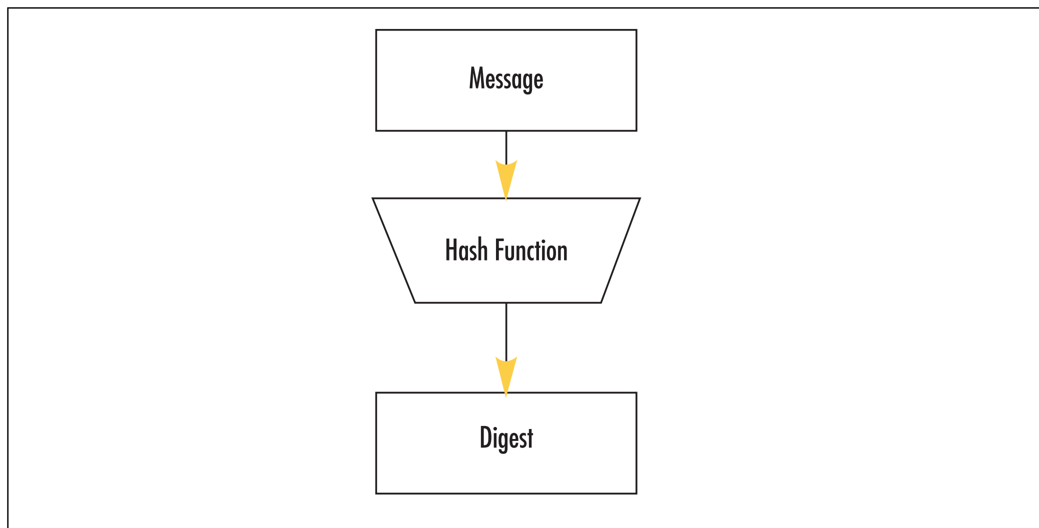
The Advanced Encryption Standard is the official NIST recommended block cipher. Its adoption is highly widespread, as it is the direct replacement for the aging and much slower Data Encryption Standard (DES) block cipher.

Integrity

Integrity is the property of ensuring correctness in the absence of an actively participating adversary. That sounds more complicated than it really is. What this means in a nutshell is ensuring that a message can be delivered from point A to point B without having the meaning (or content) of the original message change in the process. Integrity is limited to the instances where adversaries are not actively trying to subvert the correctness of the delivery.

Integrity is usually accomplished using cryptographic *one-way hash functions*. These functions accept as an input an arbitrary length message and produce a fixed size *message digest*. The message digest, or *digest* for short, usually ranging in sizes from 160 to 512 bits, is meant to be a representative of the message. That is, given a message and a matching digest, one could presume that outside the possibility of an active attacker the message has been delivered intact. Hash algorithms are designed to have various other interesting properties such as being one-way and collision resistance (Figure 1.2).

Figure 1.2 Block Diagram of a One-Way Hash Function



Hashes are designed to be one-way primarily because they are used as methods of achieving password-based authenticators. This implies that given a message digest, you cannot compute the input that created that digest in a feasible (less than exponential) amount of time. Being one-way is also required for various algorithms such as the Hash Message Authentication Code (see Chapter 5, "Hash Functions") algorithm to be secure.

Hashes are also required to be collision resistant in two significant manners. First, a hash must be a pre-image resistant against a fixed target (Figure 1.3). That is, given some value y it is hard to find a message M such that $hash(M) = y$. The second form of resistance, often cited

as 2nd pre-image resistance (Figure 1.4) is the inability to find two messages *M1* (given) and *M2* (chosen at random) such that *hash (M1) = hash(M2)*. Together, these imply collision resistance.

Figure 1.3 Pre-Image Collision Resistance

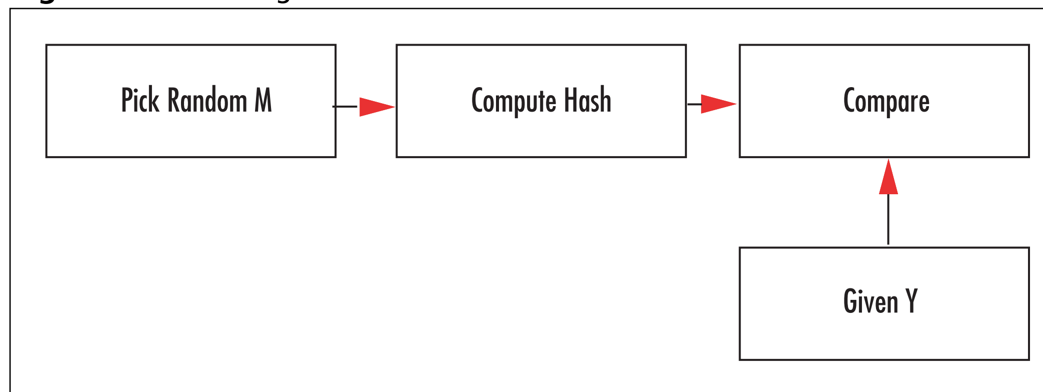
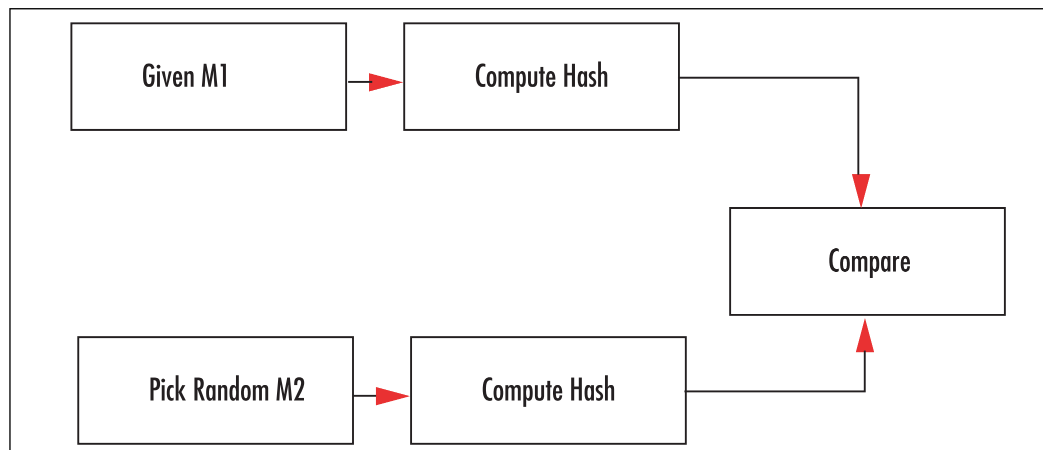


Figure 1.4 2nd Pre-Image Collision Resistance



Hashes are not keyed algorithms, which means there is no secret information to which attackers would not be privy in the process of the algorithms workflow. If you can compute the message digest of a public message, so can they. For this reason, in the presence of an attacker the integrity of a message cannot be determined.

Even in light of this pitfall, they are still used widely in computing. For example, most online Linux and BSD distributions provide a digest from programs such as md5sum as part of their file manifests. Normally, as part of an update process the user will download both the file (tarball, RPM, .DEB, etc.) and the digest of the file. This is used under the assumption

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 8

that the threat model does not include active adversaries but merely storage and distribution errors such as truncated or overwritten files.

This book discusses the popular Secure Hash Standard (SHS) SHA-1 and SHA-2 family of hash functions, which are part of the NIST portfolio of cryptographic functions. The SHA-2 family in particular is fairly attractive, as it introduces a range of hashes that produce digests from 224 to 512 bits in size. They are fairly efficient algorithms given that they require no tables or complicated instructions and are fairly easy to reproduce from specification.

WARNING!

The MD5 hash algorithm has long been considered fairly weak. Dobbertin found flaws in key components of the algorithm, and in 2005 researchers found full collisions on the function. New papers appearing in early 2006 are discussing faster and faster methods of finding collisions.

These researchers are mostly looking at 2nd pre-image collisions, but there are already methods of using these collisions against IDS and distribution systems.

It is highly recommended that developers avoid the MD5 hash function. To a certain extent, even the SHA-1 hash function should be avoided in favor of the new SHA-2 hash functions. For those following the European standards, there is also the Whirlpool hash function to choose from.

Authentication

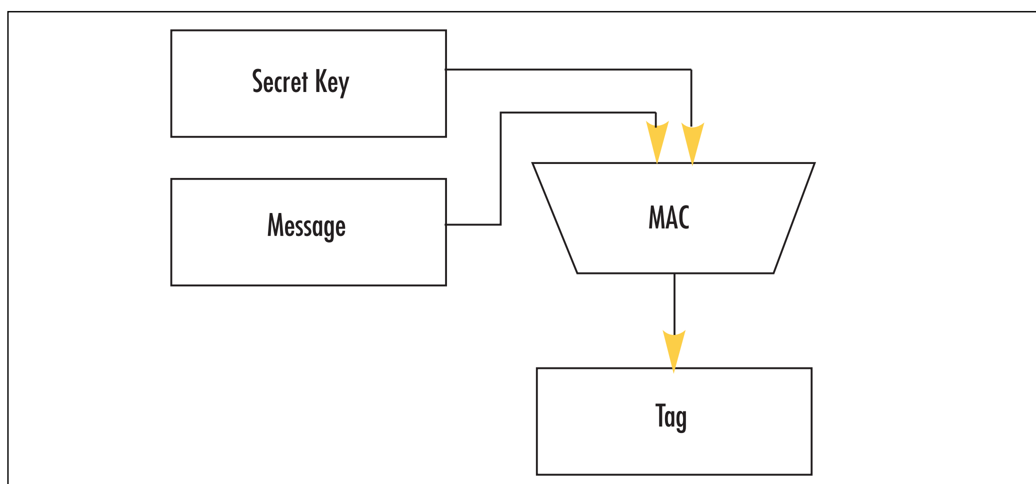
Authentication is the property of attributing an identity or representative of the integrity of a message. A classic example would be the wax seal applied to letters. The mark would typically be hard to forge at the time they were being used, and the presence of the unbroken mark would imply the documents were authentic.

Another common form of authentication would be entering a personal identification number (PIN) or password to authorize a transaction. This is not to be confused with nonrepudiation, the inability to refute agreement, or with authentication protocols as far as key agreement or establishment protocols are concerned. When we say we are authenticating a message, it means we are performing additional steps such that the recipient can verify the integrity of a message in the presence of an active adversary.

The process of key negotiation and the related subject of authenticity are a subject of public key protocols. They use much the same primitives but have different constraints and goals. An authentication algorithm is typically meant to be symmetric such that all parties can produce verifiable data. Authenticity with the quality of nonrepudiation is usually left to a single producer with many verifiers.

In the cryptographic world, these authentication algorithms are often called Message Authentication Codes (MAC), and like hash functions produce a fixed sized output called a message *tag*. The tag would be the information a verifier could use to validate a document. Unlike hash functions, the set of MAC functions requires a secret key to prevent anyone from forging tags (Figure 1.5).

Figure 1.5 Block Diagram for a MAC Function



The two most common forms of MAC algorithms are the CBC-MAC (now implemented per the OMAC1 algorithm and called CMAC in the NIST world) and the HMAC functions. The CBC-MAC (or CMAC) functions use a block cipher, while the HMAC functions use a hash function. This book covers both NIST endorsed CMAC and HMAC message authentication code algorithms.

Another method of achieving authentication is using public key algorithms such as RSA using the PKCS #1 standard or the Elliptic Curve DSA (EC-DSA or ANSI X9.62) standard. Unlike CMAC or HMAC, a public key based authenticator does not require both parties to share private information prior to communicating. Public key algorithms are therefore not limited to online transactions when dealing with random unknown parties. That is, you can sign a document, and anyone with access to your public key can verify without first communicating with you.

Public key algorithms are used in different manners than MAC algorithms and as such are discussed later in the book as a different subject (Table 1.1).

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 10

Table 1.1 Comparing CMAC, HMAC and Public Key Algorithms

Characteristic	CMAC	HMAC	RSA PKCS #1	EC-DSA
Speed	Fast	Fastest	Slowest	Slow
Complexity	Simple	Simplest	Hard	Hardest
Needs secret keys	Yes	Yes	No	No
Ease of deployment	Harder	Harder	Simple	Simple

The manner in which the authenticators are used depends on their construction. Public key based authenticators are typically used to negotiate an initial greeting on a newly opened communication medium. For example, when you first connect to that SSL enabled Web site, you are verifying that the signature on the Web site's certificate was really signed by a root signing authority. By contrast, algorithms such as CMAC and HMAC are typically used after the communication channel has been secured. They are used instead to ensure that communication traffic was delivered without tampering. Since CMAC and HMAC are much faster at processing message data, they are far more valuable for high traffic mediums.

Nonrepudiation

Nonrepudiation is the property of agreeing to adhere to an obligation. More specifically, it is the inability to refute responsibility. For example, if you take a pen and sign a (legal) contract your signature is a nonrepudiation device. You cannot later disagree to the terms of the contract or refute ever taking party to the agreement.

Nonrepudiation is much like the property of authentication in that their implementations often share much of the same primitives. For example, a public key signature can be a nonrepudiation device if only one specific party has the ability to produce signatures. For this reason, other MAC algorithms such as CMAC and HMAC cannot be nonrepudiation devices.

Nonrepudiation is a very important property of billing and accounting that is more often than not improperly addressed. For example, pen signatures on credit card receipts are rarely verified, and even when the clerk glances at the back of the card, he is probably not a handwriting expert and could not tell a trivial forgery from the real thing. Cell phones also typically use MAC algorithms as resource usage authenticators, and therefore do not have nonrepudiation qualities.

Goals in a Nutshell

Table 1.2 compares the four primary cryptographic goals.

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 11

Table 1.2 Common Cryptographic Goals

Goal	Properties
Privacy	1. Concerned with concealing the meaning of a message from unintended participants to a communication medium.
	2. Solved with symmetric key block ciphers.
	3. Recipient does not know if the message is intact.
	4. Output of a cipher is ciphertext.
Integrity	1. Concerned with the correctness of a message in transit.
	2. Assumes there is no active adversary.
	3. Solved with one-way hash functions.
	4. Output of a hash is a message digest.
Authentication	1. Concerned with the correctness of a message in transit
	2. Assumes there are active adversaries.
	3. Solved with Message Authentication Functions (MAC).
	4. Output of a MAC is a message tag.
Nonrepudiation	1. Concerned with binding transaction.
	2. Goal is to prevent a party from refuting taking party to a transaction.
	3. Solved with public key digital signatures.
	4. Output of a signature algorithm is a signature.

Asset Management

A difficult challenge when implementing cryptography is the ability to manage user assets and credentials securely and efficiently. Assets could be anything from messages and files to things such as medical information and contact lists; things the user possesses that do not specifically identify the user, or more so, are not used traditionally to identify users. On the other hand, credentials do just that. Typically, credentials are things such as usernames, passwords, PINs, two-factor authentication tokens, and RFID badges. They can also include information such as private RSA and ECC keys used to perform signatures.

Assets are by and large not managed in any particularly secure fashion. They are routinely assumed authentic and rarely privacy protected. Few programs offer integrated security features for assets, and instead assume the user will use additional tools such as encrypted file stores or manual tools such as GnuPG. Assets can also be mistaken for credentials in very real manners.

For instance, in 2005 it was possible to fly across Canada with nothing more than a credit card. Automated check-in terminals allowed the retrieval of e-tickets, and boarding

agents did not check photo identification while traveling inside Canada. The system assumed possession of the credit card meant the same person who bought the ticket was also standing at the check-in gate.

Privacy and Authentication

Two important questions concerning assets are whether the asset is private and whether it has to be intact. For example, many disk encryption users apply the tool to their entire system drive. Many system files are universally accessible as part of the install media. They are in no way private assets, and applying cryptography to them is a waste of resources. Worse, most systems rarely apply authentication tools to the files on disk (EncFS is a rare exception to the lack of authentication rule. http://encfs.sourceforge.net/), meaning that many files, including applications, that are user accessible can be modified. Usually, the worse they can accomplish is a denial of service (DoS) attack on the system, but it is entirely possible to modify a working program in a manner such that the alterations introduce flaws to the system. With authentication, the file is either readable or it is not. Alterations are simply not accepted.

Usually, if information is not meant to be public it is probably a good idea to authenticate it. This provides users with two forms of protection around their assets. Seeing this as an emerging trend, NIST and IEEE have both gone as far as to recommend combined modes (CCM and GCM, respectively) that actually perform both encryption and authentication. The modes are also of theoretical interest, as they formally reduce to the security of their inherited primitives (usually the AES block cipher). From a performance point of view, these modes are less efficient than just encryption or authentication alone. However, that is offset by the stability they offer the user.

Life of Data

Throughout the life of a particular asset or credential, it may be necessary to update, add to, or even remove parts of or the entire asset. Unlike simply creating an asset, the security implications of allowing further modifications are not trivial. Certain modes of operation are not secure if their parameters are left static. For instance, the CTR chaining mode (discussed in Chapter 4, "Advanced Encryption Standard") requires a fresh initial value (IV) whenever encrypting data. Simply re-using an existing IV, say to allow an inline modification, is entirely insecure against privacy threats. Similarly, other modes such as CCM (see Chapter 7, "Encrypt and Authenticade Modes") require a fresh Nonce value per message to assure both the privacy and authenticity of the output.

Certain data, such as medical data, must also possess a *lifespan*, which in cryptographic terms implies access control restrictions. Usually, this has been implemented with public key digital signatures that specify an expiry date. Strictly speaking, access control requires a trusted distribution party (e.g., a server) that complies with the rules set forth on the data being distributed (e.g., strictly voluntary).

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 13

The concept of in-order data flow stems from the requirement to have stateful communication. For example, if you were sending a banking request to a server, you'd like the request to be issued and accepted only once—specially if it is a bill payment! A *replay attack* arises when an attacker can re-issue events in a communication session that are accepted by the receiving party. The most trivial (initially) solution to the problem is to introduce timestamps or incremental counters into the messages that are authenticated.

The concept of timestamps within cryptography and computer science in general is also a thorny issue. For starters, what time is it? My computer likely has a different time than yours (when measured in even seconds). Things get even more complicated in online protocols where in real-time we must communicate even with skewed clocks that are out of sync and may be changing at different paces. Combined with the out-of-order nature of UDP networks, many online protocols can quickly become beastly. For these reasons, counters are more desirable than timers. However, as nice as counters sound, they are not always applicable. For example, offline protocols have no concept of a counter since the message they see is always the first message they see. There is no "second" message.

It is a good rule of thumb to include a counter inside the authentication portion of data channels and, more importantly, to check the counter. Sometimes, silent rejection of out of order messages is a better solution to just blindly allowing all traffic through regardless of order. The GCM and CCM modes have IVs (or nonces depending), which are trivial to use as both as an IV and a counter.

TIP

A simple trick with IVs or nonces in protocols such as GCM and CCM is to use a few bytes as part of a packet counter. For example, CCM accepts 13 byte nonces when the packet has fewer than 65,536 plaintext bytes. If you know you will have fewer than 4 billion packets, you can use the first four bytes as a packet counter.

For more information, see Chapter 7.

Common Wisdom

There is a common wisdom among cryptographers that security is best left to cryptographers; that the discussion of cryptographic algorithms will lead people to immediately use them incorrectly. In particular, Bruce Schneier wrote in the abstract of his *Secret and Lies* text:

> I have written this book partly to correct a mistake.

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 14

> Seven years ago, I wrote another book: *Applied Cryptography*. In it, I described a mathematical utopia: algorithms that would keep your deepest secrets safe for millennia, protocols that could perform the most fantastical electronic interactions—unregulated gambling, undetectable authentication, anonymous cash—safely and securely. In my vision, cryptography was the great technological equalizer; anyone with a cheap (and getting cheaper every year) computer could have the same security as the largest government. In the second edition of the same book, written two years later, I went so far as to write, "It is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics."
>
> *Abstract. Secret and Lies*, Bruce Schneier

In this quote, he's talking abstractly about the concept that cryptography as a whole, on its own, cannot make us "safe." This is in practice perhaps at least a little true. There are many instances where perfectly valid cryptography has been circumvented by user error or physical attacks (card skimming, for example). However, the notion that the distribution of cryptographic knowledge to the layperson is somehow something to be ashamed of or in anyway avoid is just not valid.

While relying solely on security experts is likely a surefire method of obtaining secure systems, it also is entirely impractical and inefficient. One of the major themes and goals of this book is to dispel the notion that cryptography is harder (or needs to be harder) than it really is. Often, a clear understanding of your threats can lead to efficient and easily designed cryptosystems that address the security needs. There are already many seemingly secure products available in which the developers performed just enough research to find the right tools for the job—software libraries, algorithms, or implementations—and a method of using them that is secure for their task at hand.

Keep in mind what this book is meant to address. We are using standard algorithms that already have been designed by cryptographers and finding out how to use them properly. These are two very different tasks. First, a very strong background in cryptography and mathematics is required. Second, all we have to understand is what they are meant to solve, why they are the way they are, and how not to use them.

Upon a recent visit to a video game networking development team, we inspected their cryptosystem that lies behind the scenes away from the end user. It was trivial to spot several things we would personally have designed differently. After a few hours of staring at their design, however, we could not really find anything blatantly wrong with it. They clearly had a threat model in mind, and designed something to work within their computing power limitations that also addressed the model. In short, they designed a working system that allows their product to be effective in the wild. Not one developer on the team studied cryptography or pursued it as a hobby.

It is true that simply knowing you need security and knowing that algorithms exist can lead to fatally inadequate results. This is, in part, why this book exists: to show what algo-

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 15

rithms exist, how to implement them in a variety of manners, and the perils of their deployment and usage. A text can both address the ingredients and cooking instructions.

Developer Tools

Throughout this book, we make use of readily access tools such as the GNU Compiler Collection C compiler (GCC) for the X86 32-bit and 64-bit platforms, and ARMv4 processors. They were chosen as they are highly professional, freely accessible, and provide intrinsic value for the reader.

The various algorithms presented in this book are implemented in portable C for the most part to allow the listings to have the greatest and widest audience possible. The listings are complete C routines the reader can take away immediately, and will also be available on the companion Web site. Where appropriate, assembler listings as generated by the C compiler are analyzed. It is most ideal for the reader to be familiar with AT&T style X86 assemblers and ARM style ARMv4 assemblers.

For most algorithms or problems, there are multiple implementation techniques possible. Multiplication, for instance, has three practical underlying algorithms and various actual implementation techniques each. Where appropriate, the various configurations are compared for size and efficiency on the listed platforms. The reader is certainly encouraged to benchmark and compare the configurations on other platforms not studied in this book.

Some algorithms also have security trade-offs. AES, for instance, is fastest when using lookup tables. In such a configuration, it is also possible to leak *Side Channel* information (discussed in Chapter 4). This book explores variations that seek to minimize such leaks. The analysis of these implementations is tied tightly to the design of modern processors, in particular those with data caches. The reader is encouraged to become familiar with how, at the very least, X86 processors such as the Intel Pentium 4 and AMD Athlon64 operate a block level.

Occasionally, the book makes reference to existing works of source code such as TomsFastMath and LibTomCrypt. These are public domain open source libraries written in C and used throughout industry in platforms as small as network sensors to as large as enterprise servers. While the code listings in this book are independent of the libraries, the reader is encouraged to seek out the libraries, study their design, and even use them en lieu of re-inventing the wheel. That is not to say you shouldn't try to implement the algorithms yourself; instead, where the circumstances permit the use of released source can speed product development and shorten testing cycles. Where the users are encouraged to "roll their own" implementations would be when such libraries do not fit within project constraints. The projects are all available on the Internet at the www.libtomcrypt.com Web site.

Timing data on the X86 processors is gathered with the RDTSC instruction, which provides cycle accurate timing data. The reader is encouraged to become familiar with this instruction and its use.

Summary

The development of professional cryptosystems as we shall learn is not a highly complicated or exclusive practice. By clearly defining a threat model that encompasses the systems' points of exposures, one begins to understand how cryptography plays a role in the security of the product. Cryptography is only the most difficult to approach when one does not understand how to seek out vulnerabilities.

Construction of a threat model from all use of the use cases can be considered at least partially completed when the questions "Does this address privacy?" and "Does this address authenticity?" are answered with either a "yes" or "does not apply."

Knowing where the faults lay is only one of the first steps to a secure cryptosystem. Next, one must determine what cryptography has to offer toward a solution to the problem—be it block ciphers, hashes, random number generators, public key cryptography, message authentication codes or challenge response systems. If you design a system that requires an RSA public key and the user does not have one, it will obviously not work. Credential and asset management are integral portions of the cryptosystem design. They determine what cryptographic algorithms are appropriate, and what you must protect.

The runtime constraints of a product determine available space (memory) in which program code and data can reside, and the processing power available for a given task. These constraints more often than not play a pivotal role in the selection of algorithms and the subsequent design of the protocols used in the cryptosystem. If you are CPU bound, for instance, ECC may be more suitable over RSA, and in some instances, no public key cryptography is practical.

All of these design parameters—from the nature of the program to the assets and credentials the users have access to and finally to the runtime environment—must be constantly juggled when designing a cryptosystem. It is the role of the security engineer to perform this juggle and design the appropriate solution. This book addresses their needs by showing the what, how, and why of cryptographic algorithms.

Organization

The book is organized to group problems categories by chapter. In this manner, each chapter is a complete treatment of its respective areas of cryptography as far as developers are concerned. This chapter serves as a quick introduction to the subject of cryptography in general. Readers are encouraged to read other texts, such as *Applied Cryptography*, *Practical Cryptography*, or *The Handbook of Applied Cryptography* if they want a more in-depth treatment of cryptography particulars.

Chapter 2, "ASN.1 Encoding," delivers a treatment of the Abstract Syntax Notation One (ASN.1) encoding rules for data elements such as strings, binary strings, integers, dates and times, and sets and sequences. ASN.1 is used throughout public key standards as a standard method of transporting multityped data structures in multiplatform environments. ASN.1 is

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 17

also useful for generic data structures, as it is a standard and well understood set of encoding rules. For example, you could encode file headers in ASN.1 format and give your users the ability to use third-party tools to debug and modify the headers on their own. There is a significant "value add" to any project by using ASN.1 encoding over self-established standards. This chapter examines a subset of ASN.1 rules specific to common cryptographic tasks.

Chapter 3, "Random Number Generation," discusses the design and construction of standard random number generators (RNGs) such as those specified by NIST. RNGs and pseudo (or deterministic) random number generators (PRNGs or DRNGs) are vital portions of any cryptosystem, as most algorithms are randomized and require material (such as initial vectors or nonces) that is unpredictable and nonrepeating. Since PRNGs form a part of essentially all cryptosystems, it is logical to start the cryptographic discussions with them. This chapter explores various PRNG constructions, how to initialize them, maintain them, and various hazards to avoid. Readers are encouraged to take the same philosophy to their designs in the field. Always addressing where their "random bits" will come from first is important.

Chapter 4, "Advanced Encryption Standard," discusses the AES block cipher design, implementation tradeoffs, side channel hazards, and modes of use. The chapter provides only a cursory glance at the AES design, concentrating more on the key design elements important to implementers and how to exploit them in various tradeoff conditions. The data cache side channel attack of Bernstein is covered here as a design hazard. The chapter concludes with the treatment of CBC and CTR modes of use for the AES cipher, specifically concentrating on what problems the modes are useful for, how to initialize them, and their respective use hazards.

Chapter 5, "Hash Functions," discusses the NIST SHA-1 and SHA-2 series of one-way hash functions. The designs are covered first, followed by implementation tradeoffs. The chapter discusses collision resistance, provides examples of exploits, and concludes with known incorrect usage patterns.

Chapter 6, "Message Authentication Code Algorithms," discusses the HMAC and CMAC Message Authentication Code (MAC) algorithms, which are constructed from hash and cipher functions, respectively. Each mode is presented in turn, covering the design, implementation tradeoffs, goals, and usage hazards. Particular attention is paid to replay prevention using both counters and timers to address both online and offline scenarios.

Chapter 7, "Encrypt and Authenticate Modes," discusses the IEEE and NIST encrypt and authenticate modes GCM and CCM, respectively. Both modes introduce new concepts to cryptographic functions, which is where the chapter begins. In particular, it introduces the concept of "additional authentication data," which is message data to be authenticated but not encrypted. This adds a new dimension to the use of cryptographic algorithms. The designs of both GCM and CCM are broken down in turn. GCM, in particular, due to its raw mathematical elements possesses efficient table-driven implementations that are explored. Like the MAC chapter, focus is given to the concept of replay attacks, and initialization techniques are explored in depth. The GCM and LRW modes are related in that

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 18

share a particular multiplication. The reader is encouraged to first read the treatment of the LRW mode in Chapter 4 before reading this chapter.

Chapter 8, "Large Integer Arithmatic," discusses the techniques behind manipulating large integers such as those used in public key algorithms. It focuses on primarily the bottleneck algorithms developers will face in their routine tasks. The reader is encouraged to read the supplementary "Multi-Precision Math" text available on the Web site to obtain a more in-depth treatment. The chapter focuses mainly on fast multiplication, squaring, reduction and exponentiation, and the various practical tradeoffs. Code size and performance comparisons highlight the chapter, providing the readers with an effective guide to code design for their runtime constraints. This chapter lightly touches on various topics in number theory sufficient to successfully navigate the ninth chapter.

Chapter 9, "Public Key Algorithms," introduces public key cryptography. First, the RSA algorithm and its related PKCS #1 padding schemes are presented. The reader is introduced to various timing attacks and their respective counter-measures. The ECC public key algorithms EC-DH and EC-DSA are subsequently discussed. They'll make use of the NIST elliptic curves while exploring the ANSI X9.62 and X9.63 standards. The chapter introduces new math in the form of various elliptic curve point multipliers, each with unique performance tradeoffs. The reader is encouraged to read the text "Guide to Elliptic Curve Cryptography" to obtain a deeper understanding of the mathematics behind elliptic curve math.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

FAQ

Q: When should the development of a threat model begin?

A: Usually alongside the project design itself. However, concerns such as runtime constraints may not be known in advance, limiting the accuracy of the threat model solutions.

Q: Is there any benefit to rolling our own cryptographic algorithms in place of using algorithms such as AES or SHA-2?

A: Usually not, as far as security is concerned. It is entirely possible to make a more efficient algorithm for a given platform. Such design decisions should be avoided, as they remove the product from the realm of standards compliance and into the skeptic category. Loosely speaking, you can also limit your liability by using best practices, which include the use of standard algorithms.

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 19

Q: What is certified cryptography? FIPS certification?

A: FIPS certification (see http://csrc.nist.gov/cryptval/) is a process in which a binary or physical implementation of specific algorithms is placed through FIPS licensed validation centers, the result of which is either a pass (and certificate) or failure for specific instance of the implementation. You cannot certify a design or source code. There are various levels of certification depending on how resistant to tampering the product desires to be: level one being a known answer battery, and level four being a physical audit for side channels.

Q: Where can I find the LibTomCrypt and TomsFastMath projects? What platforms do they work with? What is the license?

A: They are hosted at www.libtomcrypt.com. They build with MSVC and the GNU CC compilers, and are supported on all 32- and 64-bit platforms. Both projects are released as public domain and are free for all purposes.

404_CRYPTO_01.qxd 10/27/06 4:40 PM Page 20