

Cryptographic Design Vulnerabilities



Strong cryptography is very powerful when it is done right, but it is not a panacea. Focusing on cryptographic algorithms while ignoring other aspects of security is like defending your house not by building a fence around it, but by putting an immense stake in the ground and hoping that your adversary runs right into it.

Bruce Schneier
Counterpane
Systems

Popular magazines often describe cryptography products in terms of algorithms and key lengths. These security techniques make good headlines. They can be explained in a few words and they're easy to compare with one another. We've seen statements like "128-bit keys mean strong security, while 40-bit keys are weak" or "triple-DES is much stronger than single DES" or even "2,048-bit RSA is better than 1,024-bit RSA."

Unfortunately, cryptography isn't so simple: Longer keys *do not* guarantee more security.

Compare a cryptographic algorithm to the lock on your front door. Most door locks have four metal pins, each of which can be in one of 10 positions. A metal key sets the pins in a particular configuration. If the key aligns them all correctly, the lock opens. So there are only 10,000 possible keys, and a burglar willing to try all 10,000 is guaranteed to break into your house.

But an improved lock with 10 pins—making 10 billion possible keys—probably won't make your house more secure. Burglars don't try every possible key (the equivalent of a brute-force attack); most aren't clever enough to pick the lock (the equivalent of a cryptographic attack). No, they smash windows, kick in doors, disguise themselves as police, and rob keyholders at gunpoint. One ring of art thieves in California defeated home security systems by taking a chainsaw to the house walls. Better locks can't prevent these attacks.

Strong cryptography is very powerful when it is done right, but it is not a panacea. Focusing on cryptographic algorithms while ignoring other aspects of security is like defending your house not by building a fence around it, but by putting an immense stake in the ground and hoping that your adversary runs right into it. Smart attackers will just go around the algorithms.

Counterpane Systems has spent years designing, analyzing, and breaking cryptographic systems. While we do research on published algorithms and protocols, most of our work examines actual products. We've designed and analyzed systems that protect privacy, ensure confidentiality, provide fairness, and facilitate

commerce. We've worked with software, stand-alone hardware, and everything in between. We've broken our share of algorithms, but we can almost always find attacks that bypass the algorithms altogether.

We don't have to try every possible key or even find flaws in the algorithms. We exploit errors in design, errors in implementation, and errors in installation. Sometimes we invent a new trick to break a system, but most of the time we exploit the same old mistakes that designers make over and over again. This article conveys some of the lessons we've learned.

ATTACKS AGAINST DESIGN

A cryptographic system can only be as strong as the encryption algorithms, digital signature algorithms, one-way hash functions, and message authentication codes it relies on. In other words, break any of them, and you've broken the system. And just as it's possible to build a weak structure using strong materials, it's possible to build a weak cryptographic system using strong algorithms and protocols.

A great many systems "void the warranty" of their cryptography by using it improperly: They fail to check the size of values, reuse random parameters that should never be reused, and so on. Encryption algorithms don't necessarily provide data integrity. Key exchange protocols don't necessarily ensure that both parties receive the same key.

Some—not all—systems that use related cryptographic keys can be broken, even though each individual key is secure. Security is a lot more than plugging in an algorithm and expecting the system to work. Even good engineers, well-known companies, and lots of effort are no guarantee of robust implementation. Cryptographic weaknesses discovered in the Code Division Multiple Access (CDMA) and Global System for Mobile (GSM) communications cellular encryption algorithms and in the Microsoft Point-to-Point Tunneling Protocol (PPTP) illustrate that. In PPTP, for example, we found the strong RC4 algorithm used in a mode that almost completely negated its security.

The Global Public Key Infrastructure: Terms and Concepts

Andrew Csinger, Xcert Software
Keng Siau, University of Nebraska, Lincoln

In one sense, you're not a citizen of your country unless you can prove your nationality. And until you're part of the *global public key infrastructure* (GPKI), you're not an authentic citizen of the global electronic community.

Public and private keys

Public key cryptography involves two keys—a private key and a public key—that are mathematically related so that a message encrypted with one can be decrypted only with the other. It is extremely difficult—if not impossible—to determine the value of one key by examining the other.

The public key is often called the encryption key. To send a message to Jack—so that only Jack can read it—Jill would use Jack's public key. Jack would then use his private key to decrypt the message. Were Jack to send a reply, he would use Jill's public key, and Jill would use her private key to decrypt it.

In this way, public key cryptography helps ensure privacy.

Digests

Senders can also digitally sign their messages so that recipients have more assurance that the message is authentic. First, a user creates a unique message fingerprint—or *digest*—using a mathematical hash function. The result of encrypting the digest with a private key is a signature, which is sent with the message.

The receiver can decrypt the message and recreate the digest using the same hash function. Decrypting the signature with the sender's public key produces the original digest. If the digests match, the receiver can be certain that the message was actually sent by the signer and hasn't been altered in transit.

In this way, public key cryptography ensures both authenticity and integrity. It also ensures that the sender can't later deny having sent the message.

Public key certificates

A public key *certificate* is a digital document that irrevocably binds a person's identity to a public key. Certificates can be used as part of a process to sign digital documents in a legally binding manner or to guarantee that a message is communicated only to intended parties. Like digital signatures, certificates

help guarantee an individual's identity.

A *certification authority* (CA) creates certificates. A CA can be a service run by or on behalf of a community of interest that decides who should have membership privileges in the community. A CA can also be a government body that issues certificates to the public in conjunction with legislation governing the interpretation of digital signatures.

Communities of interest

A *community of interest* (COI) is like a country or even a country club. You're either a member or you're not. Unlike countries, communities of interest know no geopolitical boundaries. In fact, they know no fixed boundaries at all.

For example, a magazine's readership is a COI. If a magazine were to make its editorial material available on the Internet only to that readership, it becomes a true COI.

Shortly after the advent of certificate technology, it became clear that communication among CAs is critical. Now, a COI can decide whether to honor the certificates issued by the CA of another COI. Each COI can honor outside certificates in any way it chooses, from considering the certificates equivalent to its own, to restricting access to a subset of its

Random-number generators are another place where cryptographic systems often break. Good random-number generators are hard to design because their security often depends on the particulars of the hardware and software.^{1,2} Their cryptography may be strong, but if the random-number generator produces weak keys, the system is much easier to break. Some products use secure random-number generators, but they don't use enough randomness to make the cryptography secure. One of the most surprising results in this area is that specific random-number generators may be secure for one purpose but insecure for another.

Other attacks look at interactions between individually secure cryptographic protocols.³ Given a secure protocol, it is sometimes possible to build another secure protocol that will break the first if both are used with the same keys on the same device. Given the proliferation of diverse security standards using the same infrastructure, this kind of interaction failure is potentially very dangerous.

ATTACKS AGAINST IMPLEMENTATION

Many systems fail because of mistakes in implementation. Some systems don't ensure that plaintext is destroyed after it's encrypted. Other systems use temporary files to protect against data loss during a system crash or use virtual memory to increase the available memory. These features can accidentally leave plaintext lying around on the hard drive.

Buffer overflows, secrets not erased properly, and

poor error checking and recovery are all examples of implementation weaknesses that could be exploitable. In extreme cases, the OS can leave the keys on the hard drive. One product by a large software company uses a special window for password input. The password remains in the window's memory even after it is closed. It doesn't matter how good that product's cryptography is; we broke it through the user interface.

Other systems fall to more subtle vulnerabilities. Sometimes the same data is encrypted with two different keys, one strong and one weak. Other systems use master keys and then one-time session keys. Still others can be broken using partial information about different keys. And some systems use inadequate protection mechanisms for the master keys, mistakenly relying on the security of the session keys. It's vital to secure all possible ways to learn a key, not just the most obvious ones.

Electronic commerce systems often make implementation trade-offs to enhance usability. There are many subtle vulnerabilities here, when designers don't think through the security implications of their trade-offs. Doing account reconciliation only once per day, for example, might be easier, but what kind of damage can an attacker do in a few hours? Can audit mechanisms be flooded to hide the identity of an attacker? Some systems record compromised keys on hotlists; attacks against these hotlists can be very fruitful. Other systems can be broken through replay attacks—by reusing old messages or parts of old messages—to fool various parties.

information space. The restriction level is entirely up to each COI and its CA.

Unknown users

Even if a COI has never seen a particular user before, it can decide to grant access on the basis of the certificate's signature. The COI essentially says, "I don't know you, but I know and trust your CA. On the basis of that trust, I'm willing to give you access to this information."

What happens when someone with a certificate from a heretofore unknown CA approaches the boundary of a COI? The local COI has several choices. It can simply deny access and say, in effect, "Go away. I don't know you or the guy who signed your certificate."

Alternatively, the COI's defenses can try to authenticate the user. One way of doing this is to search the GPKI to see how that CA stands in relation to CAs that might be known to the COI. For example, if the unknown CA is a subsidiary of another CA known to the COI, then the COI has some basis for deciding to trust the presented certificate.

Another way of addressing this issue is to have an implicit hierarchy of CAs implied by the order of multiple signa-

tures on the certificates of users. When presented at the gates of a COI, the certificate's signatures can be checked one by one, in order, until the COI finds a signature from a CA that it already knows.

Certificate status and other attributes

For some transactions, it is not enough to rely merely on the signature of the issuing CA. When sensitive data is being accessed, it may be necessary to determine the validity of the certificate at the time of the transaction. For this reason, there are a variety of certificate status mechanisms.

The older approach—known as a *certificate revocation list* (CRL)—is maintained by a CA for the certificates it has issued. In this paradigm, certificates issued by a CA are considered valid unless they appear on the CRL. This approach mirrors the way credit card transactions were processed some decades ago and suffers from the same drawbacks.

A newer approach—still under standards development—relies on networked directory services to provide real-time certificate status information via an *online certificate status protocol* (OCSP). In this paradigm, certificates issued by a CA are assumed to be invalid until their status is

retrieved from the directory maintained by the CA. One of the advantages of an OCSP model is that it can be extended to provide information about other user attributes, like credit card numbers or home addresses.

Andrew Csinger is founder of Xcert Software, an early entrant in the PKI technology marketplace. He is vice president of electronic commerce at Group Telecom, where he operates the first and only certificate authority to be licensed under Washington State Digital Signature legislation. Csinger received an MS and a PhD in computer science from the University of British Columbia. Contact him at csinger@xcert.com.

Keng Siau is an assistant professor in the Department of Management at the University of Nebraska, Lincoln. Siau received an MS in information and computer science from the National University of Singapore and a PhD in business administration from the University of British Columbia. Contact him at klsiau@unlinfo.unl.edu.

Systems that allow old keys to be recovered in an emergency, *key escrow* systems, provide another area to attack.⁴ Good cryptographic systems are designed so that the keys exist for as short a period of time as possible; key recovery often negates any security benefit by forcing keys to exist long after they are useful. Furthermore, key-recovery databases themselves become sources of vulnerability and have to be designed and implemented securely. In some published systems, flaws in the key-recovery database could allow criminals to commit fraud and then frame legitimate users.

ATTACKS AGAINST HARDWARE

Some systems, particularly commerce systems, rely on "secure perimeter" tamper-resistant hardware such as smart cards, electronic wallets, and dongles. These systems are based on the notion that the secrets inside the secure perimeter cannot be manipulated by those not authorized access. While hardware security is an important component in many secure systems, it is prudent to distrust systems whose security rests solely on assumptions about tamper resistance.

Most tamper-resistance techniques do not work, and tools for defeating tamper resistance are getting better all the time.^{5,6} When designing systems that use tamper resistance, it is important to build in complementary security mechanisms, just in case the tamper resistance fails. It is also important to make sure that the value of the data being protected is much less than the estimated cost to defeat the tamper resistance. The

required physical protections for a system designed to meter public-transportation access, for example, are much less than those for a system designed to trade financial portfolios.

The "timing attack" made a big press splash in 1995: RSA private keys could be recovered by measuring the relative times cryptographic operations took.⁷ The attack has been successfully implemented against smart cards and other security tokens and against electronic commerce servers across the Internet. Cryptographers have generalized these methods to include attacks on a system by measuring power consumption, radiation emissions, and other "side channels," and they have implemented them against a variety of public-key and symmetric algorithms in "secure" tokens.³

Related research has looked at fault analysis, a method that deliberately introduces faults into cryptographic processors in order to determine secret keys. These attacks are almost biological in nature; they look at the cryptographic system as a complex object that responds to stimuli, and not just as a mathematical equation. The effects of this kind of attack can be devastating.

ATTACKS AGAINST TRUST MODELS

We direct many of our more interesting attacks against the underlying trust model of the system: who or what in the system is trusted, in what way, and to what extent. Simple systems—like hard-drive encryption programs or telephone privacy products—have simple trust models. Complex systems—like electronic

Building a secure cryptographic system is easy to do badly and very difficult to do well. Unfortunately, most people can't tell the difference.

commerce systems or multiuser e-mail security programs—have complex (and subtle) trust models involving many parties.

An e-mail program might use uncrackable cryptography for the messages, but unless the keys are certified by a trusted source (and unless that certification can be verified in real time) the system is still vulnerable. Some commerce systems can be broken by a merchant and a customer colluding, or by two different customers colluding. Other systems make implicit assumptions about security infrastructures, but don't bother to check that those assumptions are actually true. If the trust model isn't documented, then an engineer can unknowingly change it in product development and compromise security.

Many software systems make poor trust assumptions about the computers they run on; they assume the desktop is secure. These programs can often be broken by Trojan horse software that sniffs passwords, reads plaintext, or otherwise circumvents security measures. Systems working across computer networks have to worry about security flaws resulting from the network protocols. Computers that are attached to the Internet can also be vulnerable.

Again, the cryptography may be irrelevant if it can be circumvented through network insecurity. And no software is secure against reverse-engineering. Often, a system will be designed with one trust model in mind and implemented with another. Decisions made in the design process might be completely ignored when it comes time to sell it to customers. A system that is secure when the operators are trusted and the computers are completely under the control of the company using the system may not be secure when the operators are temps hired at just over minimum wage and the computers are untrusted.

Good trust models work even if some of the trust assumptions turn out to be wrong.

ATTACKS ON USERS

Even when a system is secure if used properly, its users can subvert its security by accident—especially if the system isn't designed very well.⁸ The classic example of this is the user who gives a password to coworkers so they can fix some problem when he's out of the office. So-called "social-engineering" attacks can often yield better results than months of cryptanalysis.⁹

Users may not report missing smart cards for a few days in case they are just misplaced. They may not carefully check the name on a digital certificate. They may reuse their secure passwords on other, insecure systems. They may not change their software's default weak security settings. Good system design can't fix all these social problems, but it can help avoid many of them.

Many systems break because they rely on user-

generated passwords. Left to themselves, people don't choose strong passwords. If they're forced to use strong passwords, they can't remember them. If the password becomes a key, it's usually much easier—and faster—to guess the password than it is to acquire the key by brute-force attack; we've seen elaborate security systems fail in this way.

Some user interfaces make the problem even worse: limiting the passwords to eight characters, converting everything to lower case, and so forth. Even passphrases can be weak: Searching through 40-character phrases is often much easier than searching through 64-bit random keys. Sometimes key-recovery systems circumvent strong session keys by using weak passwords for key recovery. The desire for fail-safe redundancy opens up a back door for attackers.

ATTACKS AGAINST FAILURE RECOVERY

Strong systems are designed to keep small security breaks from becoming big ones. Recovering the key to one file should not allow the attacker to read every file on the hard drive. Being able to forge money is a serious crime; being able to write a general program that allows anyone to forge money can destroy a currency. A hacker who reverse-engineers a smart card should only learn the secrets in that smart card, not information that will help break other smart cards in the system. In a multiuser system, knowing one person's secrets shouldn't compromise everyone else's.

Many systems have a *default to insecure* mode. If the security feature doesn't work, most people just turn it off and finish their business. This makes denial-of-service attacks very effective: If the online credit card verification system is down, merchants will default to the less-secure paper system.

Similarly, it is sometimes possible to mount a *version rollback* attack against a system after it has been revised to fix a security problem: The need for backward compatibility allows an attacker to force the protocol into an older, insecure, version.

Other systems have no ability to recover from disaster. If the security breaks, there's no way to fix it. For electronic commerce systems, which could have millions of users, this can be particularly damaging. Such systems should plan to respond to attacks and to upgrade security without having to shut the system down.

The phrase "and then the company shuts down" is never something you want to put in your business plan. Good system design considers what will happen when an attack occurs and works out ways to contain the damage and recover from the attack.

ATTACKS AGAINST CRYPTOGRAPHY

Sometimes, products even get the cryptography wrong. Some rely on proprietary encryption algorithms. Invariably, these algorithms are weak.

Cryptographers have surprising success breaking published encryption algorithms; their track record against proprietary ones is even better. Keeping the algorithm secret isn't much of an impediment to analysis anyway, because it only takes a couple of days to reverse-engineer the cryptographic algorithm from executable code.

The S/MIME 2 e-mail standard took a relatively strong design and implemented it with a weak cryptographic algorithm. The system for GSM encryption took a weak algorithm and made it weaker. And many systems just use keys that are too short.¹⁰

There are many other common cryptographic mistakes: implementations that repeat "unique" random values, digital-signature algorithms that don't properly verify parameters, or hash functions altered to defeat the very properties they're being used for. Cryptographic protocols are often used in ways unintended by the protocols' designers. For example, they are "optimized" in seemingly trivial ways that completely break their security.

PREVENTION VERSUS DETECTION

Most cryptographic systems rely on prevention as their sole means of defense: The cryptography keeps people from cheating, lying, abusing, or whatever. Defense should never be that narrow.

A strong system tries to detect abuse and to contain the effects of any attack. One of our fundamental design principles is that sooner or later every system will be successfully attacked, probably in a completely unexpected way and with unexpected consequences. It is important to be able to detect such an attack and to contain the attack to ensure it does minimal damage.

More importantly, once the attack is detected, the system needs to recover. It needs to generate and promulgate a new key pair, update the protocol, invalidate the old one, remove an untrusted node from the system, and so forth. Unfortunately, many systems don't collect enough data to provide an audit trail, or they fail to protect the data against modification.

A good audit log has to do much more than detect an attack: It must also be able to produce evidence that can convince a judge and jury of guilt.

Security designers occupy what Prussian general Carl von Clausewitz calls "the position of the interior." A good security product must defend against every possible attack, even attacks that haven't been invented yet.

Attackers, on the other hand, only need to find one security flaw in order to defeat the system. And they can cheat. They can collude, conspire, and wait for technology to give them additional tools. They can attack the system in ways the system designer never thought of.

Building a secure cryptographic system is easy to do badly and very difficult to do well. Unfortunately, most people can't tell the difference. In other areas of computer science, functionality serves to differentiate the good from the bad: A good codec will work better than a bad one, and a bad codec will look worse in feature-comparison charts.

Cryptography is different. Just because an encryption program works doesn't mean it is secure. What happens with most products is that someone reads *Applied Cryptography*, chooses an algorithm and protocol, tests it to make sure it works, and thinks the project is done. It's not.

Functionality does not equal quality, and no amount of beta testing will reveal every security flaw. Too many products are merely buzzword-compliant; they use secure cryptography but aren't secure. ❖

Bibliography

1. P. Gutmann, "Software Generation of Random Numbers for Cryptographic Purposes," *Proc. 1998 Usenix Security Symp.*, Usenix Assoc., Berkeley, Calif., 1998, pp. 243-257.
2. J. Kelsey, B. Schneier, and D. Wagner, "Protocol Interactions and the Chosen Protocol Attack," *Security Protocols, 5th Int'l Workshop*, Springer-Verlag, New York, 1996, pp. 91-104.
3. C. Hall et al., "Side-Channel Cryptanalysis of Product Ciphers," *Proc. ESORICS 98*, Springer-Verlag, New York, 1998.
4. H. Abelson et al., "The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption," *World Wide Web J.*, No. 3, 1997, pp. 241-257.
5. R. Anderson and M. Kuhn, "Tamper Resistance: A Cautionary Note," *Proc. Second Usenix Workshop Electronic Commerce*, Usenix Assoc., Berkeley, Calif., 1996, pp. 1-11.
6. J. McCormac, *European Scrambling Systems*, Baylin Publications, Boulder, Colo., 1997.
7. P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Proc. Crypto 96*, Springer-Verlag, New York, 1996, pp. 104-113.
8. R. Anderson, "Why Cryptosystems Fail," *Comm. ACM*, Nov. 1994, pp. 32-40.
9. I. Winkler, *Corporate Espionage*, Prima Publishing, Placer County, Calif., 1997.
10. M. Blaze et al., "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security," Oct. 1996, <ftp://ftp.research.att.com/dist/mab/keylength.txt>.

Bruce Schneier is president of Counterpane Systems, a cryptography and computer security consulting company. He is the author of Applied Cryptography (John Wiley & Sons, 1995) and the inventor of the Blowfish and Twofish encryption algorithms. You can subscribe to his free cryptography newsletter at <http://www.counterpane.com>.