SECURITY PITFALLS IN CRYPTOGRAPHY

V. V. Kontchevich, *PM-71*

Today all people are interested in safety of the data and reliability of programs, but the cryptography doesn't guarantee absolute reliability – it is possible to crack almost any algorithm.

But reality isn't that simple. Longer keys don't always mean more security of cryptographic product.

Hackers simply exploit errors in design, errors in implementation, and errors in installation.

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. Break any of them, and you've broken the system.

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. Specific random-number generators may be secure for one purpose but insecure for another; generalizing security analyses is dangerous.

Many systems fail because of mistakes in implementation. Some systems don't ensure that plaintext is destroyed after it's encrypted. For example, one product used a special window for password input. The password remained in the window's memory even after it was closed.

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. It too badly influences safety.

Some systems can be broken through replay attacks: reusing old messages or parts of old messages, to fool various parties.

Systems that allow old keys to be recovered in an emergency provide another area to attack. Good cryptographic systems are designed so that the keys exist for as short a period of time as possible.

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. Some systems, particularly commerce systems, rely on tamper-resistant hardware for security: smart cards, electronic wallets, dongles, etc. These systems may assume public terminals never fall into the wrong hands.

Another research has looked at fault analysis: deliberately introducing faults into cryptographic processors in order to determine the secret keys. The effects of this attack can be devastating.

Many interesting attacks are against the underlying trust model of the system: who or what in the system is trusted, in what way, and to what extent. For example, some commerce systems can be broken by a merchant and a customer colluding, or by two different customers colluding.

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. Systems working across computer networks have to worry about security flaws resulting from the network protocols.

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. The classic example of this is the user who gives his password to his co-workers so they can fix some problem when he's out of the office.

Strong systems are designed to keep small security breaks from becoming big ones. In a multi-user system, knowing one person's secrets shouldn't compromise everyone else's. Many systems have a "default to insecure mode." Other systems have no ability to recover from disaster. If the security breaks, there's no way to fix it. Good system design considers what will happen when an attack occurs, and works out ways to contain the damage and recover from the attack.

Sometimes, products even get the cryptography wrong. There are some implementations that repeat "unique" random values, digital signature algorithms that don't properly verify parameters, hash functions altered to defeat the very properties they're being used for.

Once the attack is detected, the system needs to recover: generate and promulgate a new key pair, update the protocol and invalidate the old one, remove an untrusted node from the system, etc.

All it means the following. A good security product must defend against every possible attack, even attacks that haven't been invented yet. Defense should never be that narrow. One of the 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 then to contain the attack to ensure it does minimal damage.

A. N. Dyadechko, *ELA*