Lessons Learned:
A Security Analysis of the Internet Chess Club

John Black Martin Cochran Ryan Gardner
University of Colorado
Department of Computer Science UCB 430
Boulder, CO 80309 USA
jrblack@cs.colorado.edu, Martin.Cochran@colorado.edu, ryan.gardner@colorado.edu

Abstract

The Internet Chess Club (ICC) is a popular online chess server with more than 30,000 members worldwide including various celebrities and the best chess players in the world. Although the ICC website assures its users that the security protocol used between client and server provides sufficient security for sensitive information to be transmitted (such as credit card numbers), we show this is not true. In particular we show how a passive adversary can easily read all communications with a trivial amount of computation, and how an active adversary can gain virtually unlimited powers over an ICC user. We also show simple methods for defeating the timestamping mechanism used by ICC. For each problem we uncover, we suggest repairs and draw conclusions on how to best avoid repeating these types of problems in the future.

1. Introduction

Even after several examples of how easy it is to go wrong when inventing security protocols from scratch, we still find instances of widely-deployed protocols with fundamental vulnerabilities. This paper takes a very well-known internet service, the “Internet Chess Club,” and conducts a security analysis showing that it in fact has deep security flaws. For each of the problems we uncover, we recommend ways to avoid repeating the same mistake again, and list the lessons one should take from our analysis.

BACKGROUND. Originally the the Internet Chess Server (ICS) was a free open-source server that allowed users on the Internet to play chess. The architecture was as shown in Figure 1; clients established a TCP/IP connection to a specified port on the server, and the server arranged matches between players. Each move a player made was transmitted (in the clear) to an ICS server, which would then relay that move to the opponent. The server enforced the rules of chess, recorded the position of the game after each move, adjusted the ratings of the players according to the outcome of the game, and so forth.

Serious chess players use a pair of clocks to enforce the requirement that players move in a reasonable amount of time: suppose Alice is playing Bob; at the beginning of a game, each player is allocated some number of minutes. When Alice is thinking, her time ticks down; after she moves, Bob begins thinking as his time ticks down. If either clock reaches zero before the game ends, the player who has run out of time forfeits. (We are ignoring several details here, but this is sufficient for our purposes.)

The ICS server also managed the clocks: when Alice moved, Bob would not only receive Alice’s move but also learn how much time she had taken. If either player ran overtime, the server would record the game as a loss for that player.

ICS had a number of problems. First, the server was quite buggy and would crash frequently. Also, playing fast games (say, 5 minutes per player or faster) was impractical since the network latency between client and server was charged to that player’s clock. This meant that if Alice were averaging 2 seconds round trip from her machine to the server, she would be charged 2 extra seconds, on average, for each move she made. In a fast game, this is a *very* significant disadvantage. If Alice were in Europe and the server were in the United States, fast games were simply unplayable as they presented too large a handicap for Alice.

IMPROVING THE SYSTEM. In 1992, a new manager took over running ICS and he set about fixing the many problems it had. Soon thereafter, the bugs began disappearing, and the server was more stable. Additionally, “move timestamping” was introduced.

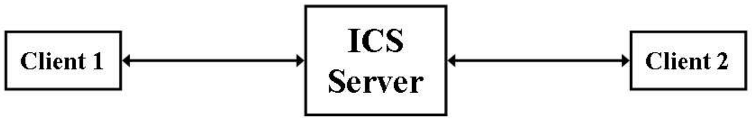

Figure 1. *The original ICS configuration: clients made a TCP/IP connection to a port on a server. The ICS server enforced the rules of chess and maintained the state of each player's clock. In this configuration, network lag would be charged to the players' clocks.*

Move timestamping is a nice idea that aims to remove the problems mentioned above regarding network lag. The basic idea is as follows (see Figure 2): when Alice receives a move from Bob, a *local* process running on Alice's machine records the time Bob's move arrived. Then, after Alice makes her move, it records the actual time Alice took; this is the time reported to the server, and this is the amount of time charged to Alice's clock. (Note that the timestamp process *could* be run on a separate machine, but then Alice is not compensated for lag between her machine and the timestamping machine.)

Of course ICS members immediately asked the obvious security question: couldn't one fake the timestamp and thereby be charged for less time than was actually used? ICS responded that two measures went toward preventing this: (1) the source code for the timestamping process would not be released, and (2) all communication to and from the server was encrypted (implying that Alice could not simply alter the outgoing packets to indicate that less time had been used). As we shall see, there are problems with both of these measures.

THE INTERNET CHESS CLUB. As a result of the improved server and the introduction of timestamping, ICS grew in popularity, and over a period of time a membership fee was introduced for those wishing full services, while still allowing guests to play for free. The server was renamed the "Internet Chess Club" (ICC) [9], and it is this service which exists today. Membership for the world's best players is free, the server is quite reliable, and the site administrators provide high-quality professional service to ICC members. As a result, most of the world's best chessmasters play on ICC, thus attracting more paying members to join as well.

Although free alternatives exist, ICC is by far the best option for serious chessplayers around the world. It boasts over 30,000 members worldwide, with hundreds of Grandmaster and International Master members. It is claimed that Madonna, Nicolas Cage, Will Smith, Sting, as well as World Chess Champion Gary Kasparov have all played chess on ICC [18, page 111]. For a fee, anyone can play against very strong masters, take lessons, subscribe to lectures, participate in simultaneous exhibitions, play in tournaments, and so forth. ICC has been written up in various newspapers and magazines, all concluding it is *the* place to play chess for the serious player. A recent book on Internet chess does likewise [18].

RESULTS. The thrust of this paper is to examine the security aspects of ICC and draw lessons from what we have learned. We analyze two distinct domains: the timestamping mechanism and the communication protocol. More specifically,

- We show a simple way to circumvent the move timestamping mechanism by modifying the binary directly or (with more effort) by decompiling the timestamp process thereby gaining source code for a compatible timestamper.
- We suggest ways to make it more difficult to achieve this kind of attack, and we give further suggestions on how to prevent or detect cheating in this way.
- We analyze the three components of the network security protocol used by ICC: key establishment, crpytographic mode of operation, and the underlying blockcipher. We show that all three are severely flawed, and exhibit attacks on each.
- We suggest simple ways of fixing the security protocols using well-known techniques and draw conclusions about the risk of using custom-written cryptographic mechanisms.

RELATED WORK. We are not the first to describe security flaws in a widely-used piece of network software. Recent examples include Goldberg and Wagner's break of the Netscape browser's random number generator [8], the break of the WEP protocol and its use of RC4 [4, 7, 19], and more recently the flaws exposed in the Diebold electronic voting system [11], flaws in Gnu Privacy Guard [13], and shortcomings of WinZip's encryption method [10].

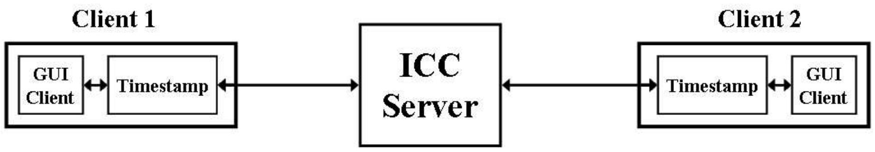

Figure 2. *The ICC configuration: clients communicate through a timestamp process which records the arrival-time of incoming moves and computes the elapsed time for a player's move before transmitting it to the server. The timestamp process can be run on a different machine, but it is most often on the local machine. Some clients have the timestamp process built-in; this is the case with the Blitzin Windows client but not with any of the Unix clients.*

THE LESSON. The lesson here is an old one: people, even very smart people, should not design their own security systems and expect them to be secure. It takes a lot of experience to get it right, and sometimes even the security experts themselves make mistakes. In each of the systems mentioned above, the security protocols were designed by non-experts and each was broken usually without a great deal of effort. For ICC, the security protocol was designed by a well-known and talented computer scientist, Professor Daniel Sleator of Carnegie Mellon University.

Our conclusion, therefore, is that development teams should never undertake the creation of custom cryptographic mechanisms for any reason. This is in spite of the fact that there exist textbooks that recommend otherwise (see, for example, [21]). Instead, well-known and well-analyzed primitives should be employed. This is a well-agreed upon tenet among security specialists, but one which has still not been widely accepted in practice.

Berkeley professor David Wagner has broken several widely-used network security protocols, including the Netscape browser in 1996 [8] and the IEEE 802.11 WEP protocol in 2001 [4]. He posted the following on the newsgroup `sci.crypt` in October 2002:

> What makes you think you can invent a good cipher if you have no expertise in the subject? Maybe you can, but it's not terribly likely. Imagine how you would react if your doctor told you "You have appendicitis, a disease that is life-threatening if not treated. We have a time-tested cure that cures 99% of all patients with no noticeable side-effects, but I'm not going to give you that: I'm going to give you a new experimental treatment my cousin dreamed up last week. No, my cousin has no medical training. No, I have no evidence that the new treatment will work, and it's never been tested or analyzed in depth–but I'm going to give it to you anyway because my cousin thinks it is good stuff." You'd find another doctor, I hope. Rational people leave medical care to the medical experts. The medical experts have a much better track record than the quacks.

STRUCTURE OF THIS PAPER. We now embark upon a systematic analysis of the ICC. We begin by studying the timestamping mechanism and not the vulnerabilities we discovered. We point out various techniques that would have made our attacks harder to mount. We then proceed to analyze the cryptographic mechanisms used by ICC and once again draw conclusions about how our resulting attacks could have effectively been averted.

2. A Security Analysis of ICC: Overview

Security on ICC is important in two respects: (1) timestamping because gaining a significant advantage means winning games, which in turn allows one to win monetary prizes in tournaments commonly offered on ICC. ICC does strive to detect players who cheat by using chess-playing computers, but they are currently incapable of detecting timestamp-cheaters. Instead there seems to be a popular misconception that timestamp is impossible to cheat [18, pp. 53–54]. (2) Since ICC encourages members to send secure information using their network protocol, the protocol needs to be correct.

We show that the encryption mechanism used by ICC is flawed in a variety of ways: there is no authentication whatsoever, and an attacker can freely flip bits of his choice in the underlying plaintext without any knowledge of the key. The blockcipher used has several differential weaknesses and is unsuitable for use as a random number generator (which is how it is used). The mode of operation is insecure and can be broken with a few bytes of known-plaintext. And, worst of all, the key exchange protocol is done insecurely, enabling a passive eavesdropper to collect all necessary key material at ICC connection-time and then record everything sent between client and server, including credit card information, ICC passwords, etc. (Despite the fact they they should not, many people probably use the same password for ICC as they have for their bank, Paypal, and other important accounts. Combine this with the public viewability of many chess players' email addresses from their profiles, and an attack could easily take over a member's Paypal account.)

An active attacker could do even more harm: a malicious man-in-the-middle could alter moves to and from the server, lie about clocks and board positions, spoof messages from the administrators, and so forth. It would probably not be hard to convince a user to reveal sensitive information if the attacker were to masquerade as an ICC administrator.

SECURITY MODELS. Note that there are distinct security models being used for each setting above: in the timestamp model, Alice herself is the adversary. She is trying to convince the server that she has used less time than she actually used. She controls the client machine and all the software running on it. This model is similar to the DRM security model[1] where achieving security is notoriously difficult. As we discuss further in Section 4, solutions to this problem are problematic.

In the communication model, where encryption is being used, Alice and the server are the communicating parties and the adversary is some outside party who is attempting to passively eavesdrop (to collect credit card numbers or listen in on Alice's interaction) or to actively corrupt Alice's session (to change her choice of move, give her a false board position, or impersonate ICC administrators in order to coax sensitive information from her). Here solutions are well-known, and we suggest some simple ones in Section 4.

3. A Dishonest Timestamp Client

ICC's attempt to avert timestamp cheaters was to release only the binary for the program and to encrypt (but not authenticate) its output. ICC personnel undoubtedly knew that it was not too hard to circumvent these precautions, but they needed a simple, cost-effective, secure solution to the problem of network lag, and this is the approach chosen. We now examine the results.

WITHHOLDING THE SOURCE. ICC chose to withhold the source for the timestamp program; this is reasonable: if the source were freely distributed, it would be a very simple matter to modify the program to cheat in arbitrary ways. Controlling the source code is a common way for many companies to attempt to retain control over their code. Of course this works only to some extent: reverse engineering a binary entails some amount of work, but for small programs it is quite reasonable. We decided to reverse engineer a piece of ICC code because it was quite easy to do so, thanks to the way Linux is supported for ICC members.

[1]DRM stands for "Digital Rights Management," a technology which attempts to prevent users from copying software, music, video, and other content.

REVERSE ENGINEERING THE LINUX CLIENT. By far the most popular client for ICC runs on Microsoft Windows and is called "Blitzin." It is about 2.15 megabytes in size, and the timestamping (and encryption) is built-in. Given that our analysis tools consisted primarily of a debugger, we opted instead to examine the Linux timestamp program. The Linux program is only 27 kilobytes and is separate from the graphical clients that use it. Moreover, the Linux binary has symbols intact. This means that program labels for static variables and function names were listed within the binary and Linux programs such as `nm` or `objdump` would list helpful names like `encrypt`, `decrypt`, `set_base_time`, and so forth. It also meant that when using our debugger of choice, `gdb`, these symbols would be listed when disassembling code or setting breakpoints. It would have made our job a good deal harder had ICC run the Linux program `strip` in order to remove symbols from the binary before distributing it.

> **Lesson #1:** If you are going to attempt to hide a program's functionality by withholding the source code, at least strip symbols from the object code.

Another attempt to make reverse engineering harder is to use a program obfuscator. Although this has been shown to be impossible in a general sense [1], game producers often use such techniques in an attempt to slow down the piracy of their products, and some attempts have been made (with mixed results) to build a theory of practical obfuscation techniques [5].

> **Lesson #2:** If you are going to attempt to hide a program's functionality, use an automated obfuscation program. (This is not a perfect solution, but has some measure of effectiveness.)

REVERSE ENGINEERING. Given the small size of the Linux timestamping client, we decided to reverse engineer it as a test to see how hard this would be. The presence of symbols and the lack of any obfuscation made the job quite doable: the task required about 65 hours of work. With our reverse-engineered client, sophisticated rules can be established for deducting time without arousing much suspicion from ICC administrators. The most natural idea is to subtract some constant amount of time from the time actually used, giving the server the appearance that the average lag between it and the client is some number of seconds greater than it really is.

LESSONS TO BE LEARNED. ICC undoubtedly knew that it was not too hard to make a dishonest timestamp client. They were faced with trying to make Internet Chess a fairer experience without going to extraordinary lengths.

In order to prevent cheating, we would have to remove control of the timestamp process from the adversary (ie, the player). This immediately leads to problems: if we move the timestamp functionality upstream (ie, toward the server), then the user pays for network lag between his machine and the timestamper. Even if a trusted ISP were to offer timestamping service (which might be useful in several contexts other than Internet Chess), it would be only a partial solution: lag from the client machine to the ISP would be charged against the player, and most likely many small ISPs would probably not offer the service. Also, it would probably not be hard for Alice to pretend she is an ISP and timestamp packets herself such that upstream routers would leave them alone.

Since rearchitecting the Internet is both infeasible and falls short of a full solution, we are faced with keeping the timestamping functionality close to the user. In order to prevent Alice from tampering with the timestamp, it seems that using secure hardware is the only real solution. The idea is to put a card into the bus of Alice's computer that computes the elapsed time (with its own clock) and uses proper encryption and authentication to produce a message for the ICC server. (The encryption is to prevent upstream viewing of an incoming move by a confederate of Alice who then relays the move to her.) The problem, of course, is that such cards cost money and requiring every ICC user to purchase such a card would likely be prohibitive.

> **Lesson #3:** For some applications where the user himself controls the machine where secure code is run, the only solution may be to employ (relatively expensive) tamper-proof hardware.

4. Cryptanalysis

In the previous section we showed how to simply defeat the timestamp security of ICC, and reached the somewhat unsatisfactory conclusion that secure hardware may be the only good solution to the problem. In this section we examine the cryptographic methods used by the ICC client and show that they are easily attacked. In this case we are able to offer straightforward and well-known ways to repair the defects.

OVERVIEW. There are two main components to the ICC cryptographic protocol: (1) key establishment and (2) encryption. Key establishment is done only once at session start-up time using a protocol between client and server which is described later. The goal of key establishment is to share two 64-bit strings called the `send_encryptor` key and the `receive_encryptor` key. These keys are used by a symmetric encryption scheme with the `send_encryptor` key of the client matching the `receive_encryptor` key of the server and vice-versa.

The encryption protocol consists of a mode of operation over a custom blockcipher. It is used after key establishment to encrypt and decrypt messages between the client and server. Note that *no* authentication is attempted in the protocol and, as we shall see, it is trivial to manipulate the plaintext because the mode is essentially a one-time pad [12]. This defect is particularly relevant to timestamping, since the main goal in installing the encryption protocol was to prevent timestamp tampering.

4.1. Key Establishment

OVERVIEW. Key establishment works as follows: at session start-up the client and server each choose a pseudo-random 64-bit string. The server and client then exchange these seed values. Then the server and client each perform a deterministic process (involving the blockcipher we will discuss shortly) which depends only on the two exchanged seeds (see Figure 3). This means that obtaining the seeds and understanding the key derivation process enables a passive eavesdropper to easily decrypt all subsequent communication. An active adversary can mount a man-in-the-middle attack or even impersonate the ICC server in order to extract information from the user.

> **Lesson #4:** For key exchange to be secure, asymmetric techniques must be used (such as Diffie-Hellman key exchange [6]), and even this needs to be authenticated. Simply exchanging keying information in the clear, as ICC does, essentially removes any hope of security against even a passive attacker.

AN ICC SNIFFER. We coded a simple "ICC sniffer" using the freely-available `pcap` library to extract packets from the network. Our sniffer extracts the two seeds as they are exchanged between client and server during the key establishment protocol and then dumps all subsequent communication to the screen. This of course requires that the sniffer understand the encryption and decryption procedures, but these were reverse engineered from the Linux timestamp client. (Extracting just the code relevant to the encryption and decryption routines required about 25 hours.) We did not write any code for mounting an active attack, though this would not be hard. Man-in-the-middle attacks can be mounted on insecure connections via standard tricks like ARP cache poisoning [16] and DNS spoofing.

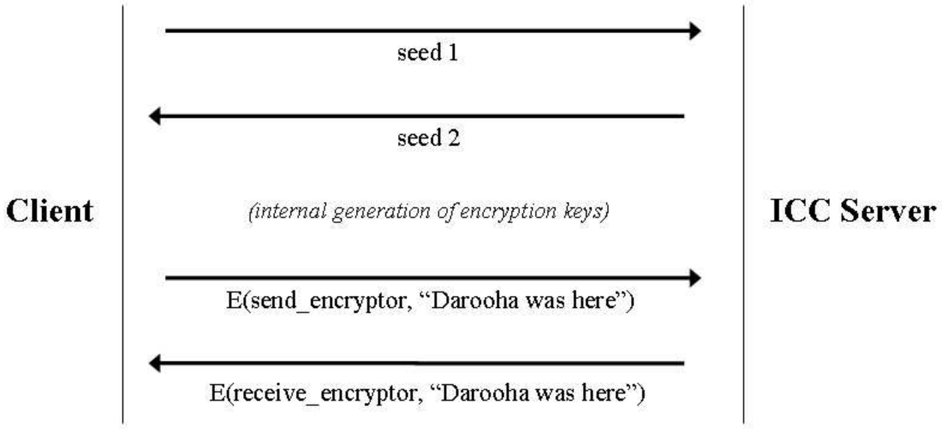
seed 1
seed 2
Client
(internal generation of encryption keys)
ICC Server
E(send_encryptor, "Darooha was here")
E(receive_encryptor, "Darooha was here")

Figure 3. *The ICC Key Establishment Protocol: two pseudo-random 64-bit strings are exchanged between client and server. A deterministic process then computes symmetric sender and receiver keys on each end. An encrypted test message "Darooha was here" is then sent using the derived keys. ("Darooha" is the ICC designer's nickname.)*

REMEDIES. Users wishing to pay online should be required to submit membership fees through a web-based payment gateway using SSL/TLS. This would at least protect credit-card information for ICC members.

In order to prevent all passive and active attacks, we must repair the protocol. The obvious solution is to use a proper key exchange based on Diffie-Hellman [6] or RSA [17]. Freely-available libraries such as OpenSSL [20] could be used to quickly insert this functionality (at the cost of expanding the size of the client binary).

Unfortunately, fixing just the key exchange protocol is insufficient. There still is no authentication, and the mode of operation and the blockcipher still have serious defects which we now describe.

4.2. The ICC Mode of Operation

OVERVIEW. The ICC Mode of Operation uses the blockcipher (described next) to produce a pseudo-random seed to two linear congruential generators (LCGs). These two generators each produce 100 bytes of output, and these bytes are XORed to form a pad which is used for encryption and decryption [12, page 21]. It is well-known that LCGs are not cryptographically strong and they should not be used to generate pads [12, pp. 170–187]. One should therefore be suspicious of a technique that XORs together two LCG outputs for use as a pad. And indeed we show that the ICC mode which employs this tack does in fact not work. We are able to recover the entire pad given about 10 bytes of pad.

THE MODE. Let s_i denote the i-th byte of a string s where we count from left-to-right starting at 0. Let $\oplus$ denote the XOR operation on same-length strings. For an n-byte message m the ICC mode produces ciphertext c by computing $c = m \oplus r$ where r is a pseudo-random string of n bytes. Each r_i is generated by XORing together two bytes output by two LCGs (see Figure 4). Specifically, $r_i = y_i \oplus z_i$ where

$$w_i = 17 w_{i-1} \bmod 2413871 \quad \text{and} \quad y_i = w_i \bmod 2^8$$

$$x_i = 3x_{i-1} + 1 \bmod 43060573 \quad \text{and} \quad z_i = x_i \bmod 2^8.$$

Every 100 bytes we reset i to 0 and re-seed w_0 and x_0 to new values generated by the blockcipher. For the purposes of our attack, these seeds can be arbitrary.

THE ATTACK. Our attack produces all 100 relevant x_i and w_i values given that 10 or so consecutive bytes of a given message are known. This seems reasonable given the consistency of the messages sent by ICC. The attack has elements of both brute force and cleverness, but mostly of the former. The attack runs in about 1.1 seconds on a laptop with an AMD XP 2400+ processor.

Note that the attack is not general: changing the constants in the LCGs to different values can make this attack computationally infeasible. We believe the general approach of XORing together the output of two LCGs cannot be right, but for the purposes of this paper we focus on the LCGs used by ICC.

We start by choosing some i such that we know $m_i \cdots m_{i+9}$. This allows us to determine $r_i \cdots r_{i+9}$. We

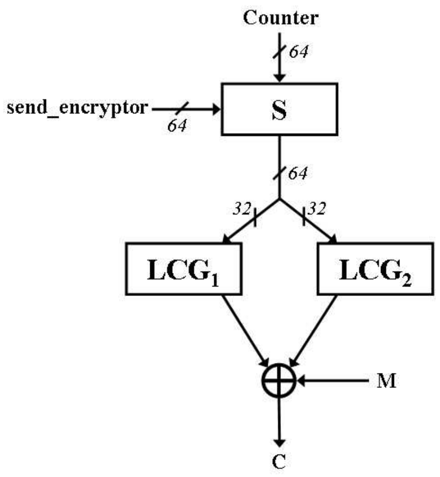

Figure 4. *The ICC Mode of Operation: 64 bits are generated by enciphering a counter under the sender's blockcipher key. The first 32 are sent to LCG$_1$ and the other 32 to LCG$_2$. The LCG's are iterated 100 times and their lowest bytes are XORed to produce an encryption pad which is XORed with the message M to produce the ciphertext C.*

will try to guess y_i outright. This means an outer loop guessing the values in $[0, 2413870]$. Upon a guess g_i of y_i, the algorithm computes $g_i, ..., g_{i+9}$ and the corresponding guesses for z_i, which we will call $h_i, ..., h_{i+9}$. Let x_j^h (resp. w_j^h) denote the 24 most significant bytes of x_j (resp. w_j) such that $x_j = 2^8 x_j^h + z_j$ (resp. $w_j = 2^8 w_j^h + y_j$). Statistically, we expect the following relation to hold for about 1 out of every 3 values of j:

$$x_j < (43060573 - 1)/3.$$

This implies that

$$z_{j+1} \equiv 3x_j + 1 \equiv 3(2^8 x_j^h + z_j) + 1 \equiv 3z_j + 1 \pmod{2^8}.$$

We will check the values $h_i, \ldots, h_{i+9}$ for this property. With 10 known values of y_i, when we find the correct values $z_i, \ldots, z_{i+9}$, we expect this relation to hold for 3 consecutive pairs (the probability that it does not hold for any pairs is less than 3%–this can be reduced with knowledge of more characters of M). When we have not found the correct values of z_i, the expected number of times this happens is much less.

When we find h_j such that $h_{j+1} \equiv 3h_j + 1 \pmod{2^8}$, we will find the next pair (h_k, h_{k+1}) such that $h_{k+1} \not\equiv 3h_k + 1 \pmod{2^8}$. This implies that $3x_k + 1 \geq 43060573$. Let us define $c = 3^{k+1-j} 2^8 x_j^h$. This implies that one of the following two cases hold:

$$h_{k+1} \equiv 3h_k + 1 + c - 43060573 \mod 2^8 \quad (1)$$

$$h_{k+1} \equiv 3h_k + 1 + c - 2(43060573) \mod 2^8 \quad (2)$$

Without loss of generality, consider that (1) has occurred. Then $h_{k+1} - 3h_k - 1 + 43060573 \equiv c \pmod{2^8}$, but $c \equiv 0 \pmod{2^8}$, so we will be able to determine between the first and second cases by examining the values of $h_{k+1} - 3h_k - 1 + 43060573 \bmod 2^8$ and $h_{k+1} - 3h_k - 1 + 2(43060573) \bmod 2^8$. If neither case held, then we can be sure that the current guesses for h_j are incorrect. Alternatively, if one of the cases held we can be sure that the guesses for h_i are correct with probability roughly $(1 - 18/2^{16})$. [2]

We know that $43060573 - 3h_k - 1 < c < 2(43060573) - 3h_k - 1$, which implies that $\sim 18000 < x_j^h < \sim 36000$. We can exhaustively check these remaining values to find the correct one.

Every 100th character will probably not be deciphered correctly. This is a technical issue having to do with the way every 100th r_i is computed. When w_i and x_i are re-seeded, they are not necessarily smaller than the moduli used in the LCGs. Thus the value $17^{-1} w_{i+1} \bmod 2413871$ is not necessarily the w_i that was used to compute r_i. The correct character can almost always be inferred from context, however.

The expected number of divisions and multiplications is about 2^{26} (the loop iterating over values in $[0, 2413870]$ that computes the g_i and h_i dominates).

[2] For purposes of a rough estimate, we consider the probability, given incorrect $h_i, \ldots, h_{i+9}$, that some pair h_j, h_{j+1} satisfies $h_{j+1} \equiv 3h_j + 1 \pmod{2^8}$ to be $9/2^8$. We also consider the probability that the pair h_k, h_{k+1} satisfies (1) or (2) to be less than $2/2^8$.

Lesson #5: There are many modes of operation that are provably-secure and provide privacy and authentication. See [3] for a survey. Though it should be mentioned that even here, implementors need to take care to avoid attacks based on implementation details such as power and timing analyses, reaction attacks and so forth. Once again, we strongly recommend a consultation with experts in network security protocols to increase assurance and reduce risk.

4.3. The ICC Blockcipher

OVERVIEW. A blockcipher is a very general cryptographic object which can be used for a multitude of purposes [12]. A blockcipher is an algorithm which takes two inputs: an n-bit input message block M and a k-bit key K, and produces an n-bit output message block C. A necessary requirement is that for any key K, if M and M' are distinct input message blocks then enciphering them yields distinct output message blocks. This is because blockciphers are often used for encryption where we must be able to decipher what we have enciphered. The most well-known blockciphers are DES [14] and AES [15].

There are instances where a blockcipher need not be invertible, however. For example, in counter-mode encryption [12, page 233] we simply fix a key and then input counter values 0, 1, 2, etc., trusting that their encipherments are pseudo-random and can be used as pads. The ICC mode of operation described above is similar: it uses a counter enciphered by the ICC blockcipher to generate seeds to two LCGs (and, as we have pointed out, this is *not* a good practice).

We now proceed to describe and analyze the ICC blockcipher. The remainder of this section assumes basic knowledge about blockcipher construction (see, for example [12]).

BLOCKCIPHER DESCRIPTION. Blockcipher S is a 16-round Feistel blockcipher [12], taking a 64-bit input and a 64-bit key. There is no pre- or post-processing prior to the Feistel rounds.

The round-function $f : \{0,1\}^{64} \times \{0,1\}^{32} \times \{0,1\}^{32} \rightarrow \{0,1\}^{32}$ takes a 64-bit key K, a 32-bit round value V, and the round number r (taken as a 32-bit integer). All arithmetic is signed and modulo 2^{32}. (In other words, computations are carried out in 32-bit signed registers with the carry disregarded). For any string X let $X[i]$ denote the i-th least significant byte of X. The function f is then

$$f(K, V, r) = \\ \mathit{stuff}[(V[0] + V[1] + K[r \bmod 8]) \bmod 256] + V^2$$

where stuff is a static table of 256 32-bit values. (The stuff table is generated once again by LCGs, but this is not relevant for the analysis we conduct below.) Note that only one byte of key K is used per round and that only the lowest 2 bytes of V are used in indexing stuff. This immediately suggests that the high bits are not affecting the round function as much as they perhaps should.

Let S be the blockcipher resulting from iterating f for 16 rounds using the Feistel construction, and let $S(K, P)$ denote running S with key K and input P.

ANALYSIS. Cipher S has serious flaws. The easiest one to spot is this: for any plaintext block P, flipping the high bit of P merely results in flipping the high bit of $S(K, P)$, independent of the key. We state this more formally by first proving a simple property of the round function:

Proposition 1 *Let $B = 2^{31}$, and notice that for a 32-bit quantity V, writing $V \oplus B$ denotes flipping the high-bit of V. Then for any values of K, V, and r, we have $f(K, V, r) = f(K, V \oplus B, r)$.*

Proof: Let K be arbitrary and fixed. Since the index value to stuff depends only on the lowest two bytes of V, clearly $V \oplus B$ will produce the same index. We therefore focus on the squaring operation.

Since arithmetic is signed, B is interpreted as -2^{31} and therefore $V \oplus B$ can be thought of as $V - 2^{31}$ where V is taken as signed integer. Squaring this quantity

$$(V - 2^{31})^2 \equiv V^2 - 2^{32}V + 2^{62} \equiv V^2 \pmod{2^{32}}.$$

Therefore flipping the high bit of V and squaring is the same as squaring V itself when working modulo 2^{32}, and therefore f produces the same value overall. ▮

This invariant propagates throughout the cipher: consider two 64-bit inputs $P = (X, Y)$ and $P^* = (X \oplus B, Y)$ where the quantities in the parentheses are 32-bit values. Because of the above invariant on f, we see that if $S(K, P) = (X', Y')$ then $S(K, P^*) = (X' \oplus B, Y')$ independent of the choice of K. In other words, flipping the most significant bit of P results in flipping the most significant bit of its ciphertext, regardless of what key is used. It is clear that $P^* = (X, Y \oplus B)$ has analogous behavior. In the language of Biham and Shamir, we have a probability 1 differential characteristic [2].

This means that S can be distinguished from a random permutation with exactly two chosen plaintexts. It also means that S is not a very good random number generator (which is the purpose it is being used for here). It can be shown that when the counter is as little as 2^{16}, bytes 0,1,4, and 5 of $S(K, 2^{16})$ will be the same as those in $S(K, 0)$ independent of K.

In short, the cipher does a poor job diffusing minor changes in plaintexts. Its “avalanche effect” is insufficient, and therefore the cipher is weak. It should not be used in any cryptographic setting.

> **Lesson #6:** There are numerous public-domain blockciphers available whose security is believed to be very strong. Most commercial implementations now use AES [15] given that it is now an internationally-recognized standard. No significant flaws have been discovered in AES since its introduction several years ago, and in this sense it is far superior to the ICC blockcipher described above.

5. Conclusion and Open Problems

We have scrutinized the security aspects of ICC and uncovered several problems. The timestamping mechanism is easily circumvented, allowing malicious users to cheat at chess by unfairly gaining time on the clock.

We have also analyzed the ICC network security protocol and shown it is flawed in numerous ways enabling passive eavesdroppers to trivially listen in on communications and enabling active adversaries to mount severe attacks on ICC users. The important lesson we may take from this is that it is very hard to devise security protocols which work. It seems that whenever a non-expert invents his own, even if he is very clever, it is often broken. This has long been a message espoused by the security community, but the battle has not yet been won.

6. Acknowledgements

We would like to thank David Wagner, Yoshi Kohno, Jing Deng, and Douglas Sicker for their comments and suggestions. Thanks also to Cindy Cohn and Jason Schultz of the Electronic Frontier Foundation for looking over the paper early on. John Black’s and Martin Cochran’s work was supported by NSF CAREER-0240000 and a gift from the Boettcher Foundation. Ryan Gardner’s work was supported by NSF REU-0244168.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2001.

[2] E. Biham and A. Shamir. *A Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[3] J. Black. Authenticated encryption, 2003. Preprint available at `www.cs.colorado.edu/~jrblack`.

[4] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: The insecurity of 802.11. In *MOBICOM*, pages 180–189. ACM, 2001.

[5] S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. White-box cryptography and an AES implementation. In H. Heys and K. Nyberg, editors, *Selected Areas in Cryptography — SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, Nov. 1976.

[7] S. R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In S. Vaudenay and A. Youssef, editors, *Selected Areas in Cryptography — SAC 2001*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[8] I. Goldberg and D. Wagner. Randomness and the Netscape browser, Jan. 1996. Dr. Dobbs Journal.

[9] Internet Chess Club. See ICC website at `www.chessclub.com`.

[10] T. Kohno. Analysis of the WinZip encryption method. In *11th ACM Conference on Computer and Communications Security–CCS 2004*. ACM Press, 2004.

[11] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy 2004*. IEEE Press, 2004.

[12] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[13] P. Q. Nguyen. Can we trust cryptographic software? Cryptographic flaws in GNU Privacy Guard v1.2.3. In *Advances in Cryptology – Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 555–570. Springer-Verlag, 2004.

[14] NIST. Data Encryption Standard (FIPS 46-2), 1988.

[15] NIST. Advanced Encryption Standard (FIPS 197), 2001.

[16] A. Ornaghi and M. Valleri. Ettercap. A program for man-in-the-middle attacks on Ethernets; see `ettercap.sourceforge.net`.

[17] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[18] R. Schmaltz. *The Complete Chess Server Guide*. Schachzentrale Rattmann, 2004. ISBN: 3-88086-180-3.

[19] A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *Proceedings of Network and Distributed System Security Symposium 2002 (NDSS’02)*, San Diego, CA, February 2002.

[20] J. Viega, M. Messier, and P. Chandra. *Network Security with OpenSSL*. O’Reilly, 2002. See also the OpenSSL website at `www.openssl.org`.

[21] Y. Zheng and S. Akhtar. *Networks for Computer Scientists and Engineers*. Oxford University Press, 2002. ISBN: 0-19-511398-5.