How to Cheat at Chess: A Security Analysis of the Internet Chess Club

J. Black [*] M. Cochran [*] R. Gardner [*]

November 15, 2004

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. Most of these are practical and inexpensive.

Keywords: Security, cryptanalysis, online gaming, computer chess.

1 Introduction

Background. In the early 1990's the Internet Chess Server (ICS) was a free open-source server that allowed users on the Internet to play chess against each other. Some users played via text-based interfaces, but a few rudimentary graphical interfaces existed as well. 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 the 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.

As players in the 19th century discovered, it can be extremely frustrating to play chess without a time control: the opponent simply refuses to move when he is losing. These days all serious chess players use a pair of clocks: 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 nuances 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.

[*]Department of Computer Science, 430 UCB, Boulder, Colorado 80309-0430 USA.
E-mail: jrblack@cs.colorado.edu, Martin.Cochran@colorado.edu, ryan.gardner@colorado.edu
WWW: www.cs.colorado.edu/∼jrblack, ucsu.colorado.edu/∼cochranm, www.rwgardner.net

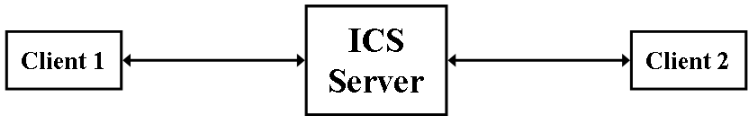


Figure 1: *The original ICS configuration: clients made a TCP/IP connection to a port on a server. The ICS server enforced the rules of chess and maintained the state of each player's clock. In this configuration, network lag would be charged to the players' clocks.*

Daniel Sleator. Daniel Sleator is a professor in the Department of Computer Science at Carnegie Mellon University, and is a well-regarded researcher in theoretical computer science. He is probably best-known for inventing Splay Trees with Tarjan [19]. In 1992, Sleator became the chief programmer for ICS, and he set about fixing the many problems it had. Players soon noticed vast improvements on ICS: the bugs were disappearing, the server was more stable. Additionally, Sleator introduced the idea of "move timestamping."

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? Sleator 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. However, several people complained that Sleator was not releasing his improved source code. It became clear that Sleator had commercial aspirations for ICS, and over a period of time he introduced a membership fee for those wishing full services, while still allowing guests to play for free. (Guests were restricted in what they were allowed to do, and these days there are very few things a guest can do other than play chess with other guests.)

Sleator renamed the server the "Internet Chess Club" (ICC) [9]. 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, many of the world's best chessmasters play on ICC, thus attracting more paying members to join as well. ICC membership is $49 per year (students pay a lower rate). Recently Sleator incorporated as "Sleator Games, Inc."

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 Champion Gary Kasparov have all played chess on ICC [18, page 111]. For a fee, anyone can play against incomprehensibly strong masters, take lessons, listen 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. We exhibit attacks in two distinct domains: the timestamping mechanism and the communication protocol. More specifically,

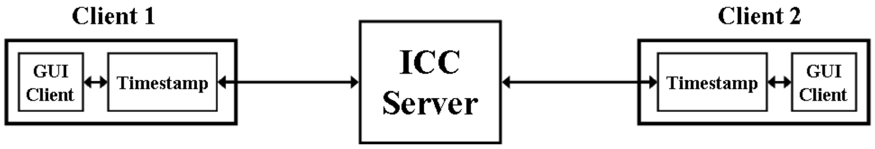


Figure 2: *The ICC configuration: clients communicate through a timestamp process which records the arrival-time of incoming moves and computes the elapsed time for a player's move before transmitting it to the server. The timestamp process can be run on a different machine, but it is most often on the 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.*

- 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 our 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, mode of operation, and the 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.

As a proof of both concepts above, we have written code to show that our attacks are easily realized with little work. Specifically, we have built a cheating timestamp client and tested it. (It was never used in a rated game or to play against registered ICC members.) And we have built an "ICC sniffer" which passively watches an ICC connection and then records all communication between client and server. (It has never been used except to eavesdrop on the authors' own connections.) We do not plan to release either piece of software.

Related Work. We are not the first to find 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, 20], 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].

Most of our methods are simple, and most attacks are original though we do employ a basic form of differential cryptanalysis [2] to analyze the ICC block cipher.

The Moral. 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. 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. Indeed, any security expert who decided to expend some time and effort could have achieved most of the results above and in this paper.

2 A Security Analysis of ICC: Overview

We now conduct a security analysis of ICC. But before embarking, it might be fair to ask "who cares?" In other words, if we show you how to fake a timestamp which lets you gain a significant advantage playing Internet Chess, doesn't that just mean you unfairly win a few games and some illegitimate rating points?

More importantly, one might question the existence of any danger in the ability of eavesdroppers to observe a few communications to and from a game server. In fact, both of these issues are significant.

Although most of the tournaments on ICC offer no awards, some do offer cash prizes: for example, the 2001 "Dos Hermanas" tournament on ICC offered a top prize of 1500 Euros (about $1,350 at that time). This in itself might be enough incentive to cheat. A cheating timestamp client means that Alice could have several additional seconds to think, without being charged for them. It means she could even have enough time to transfer Bob's move into a strong chess-playing program, even in a very fast game. This would allow Alice to use another program to generate very good moves against Bob, and the time taken to relay moves to and from this program would not cost her. The World 1-Minute Chess Champion, Roland Schmaltz, claims that one need not worry about this form of cheating in games faster than 3 minutes [18, page 54], but clearly a cheating timestamp client enables this quite easily. It would just appear that Alice had a few extra seconds of network lag.

ICC does strive to detect players who cheat by using chess-playing computers, but they currently incapable of detecting timestamp-cheaters. Instead there seems to be a popular misconception that timestamp is impossible to cheat [18, pp. 53–54].

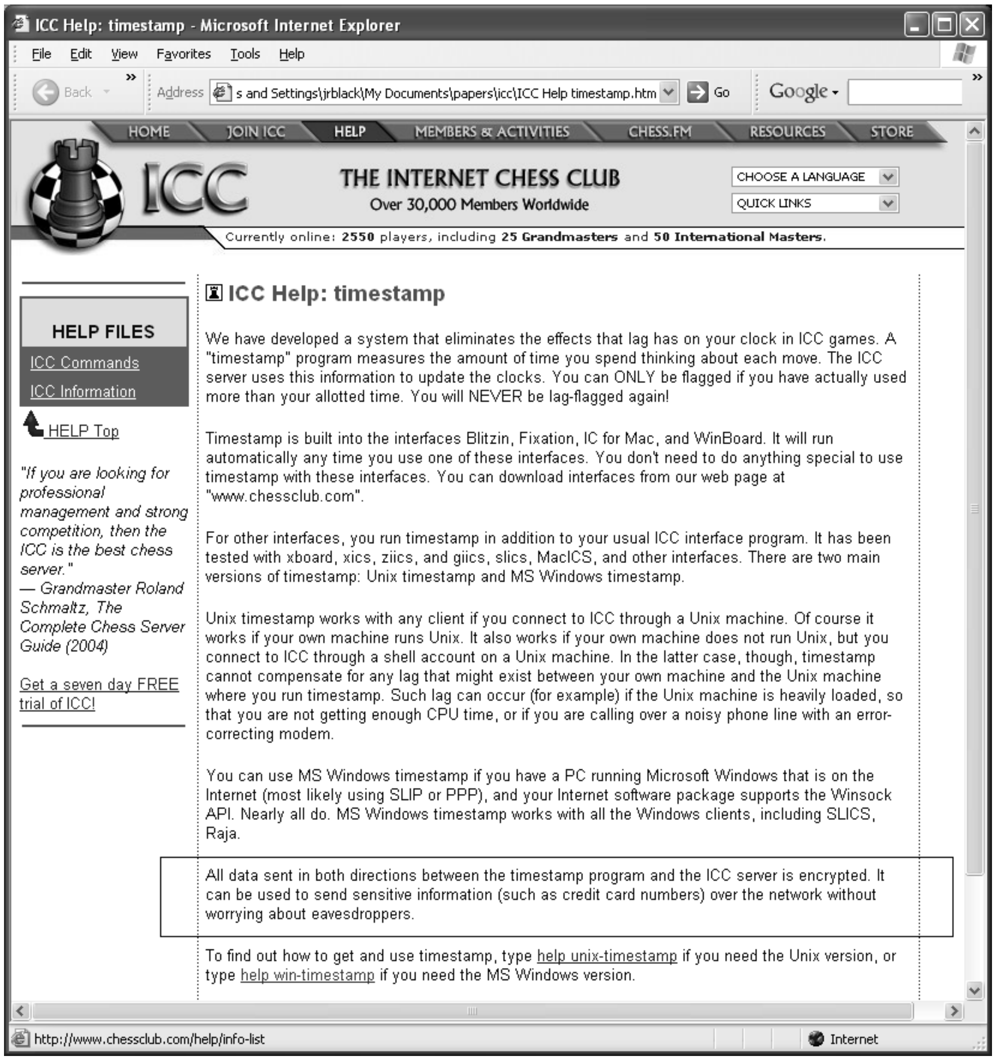
The second target of our analysis is the network security protocol used by ICC. We believe the initial reason to introduce encryption into the timestamp process was to prevent timestamp tampering. However, the presence of encryption seems to have emboldened the ICC managers to claim that secure information can be safely sent to the server. See Figure 3, which was taken from the ICC web page. Indeed, when a member wishes to extend his membership, he is presented with 3 options:

```
You have the following payment options:

  1) Give credit card information on-line now.  Your membership will
     be extended immediately.  The data will be encrypted before it is
     sent, and the credit card number will be stored on our system
     only in encrypted form, and will be read only by the ICC treasurer.
  2) Give credit card information by email, telephone, or FAX.
  3) Send credit card information, personal check, or postal money
     order by regular postal mail.

Please give your choice ("1", "2", "3", or "quit"):
--->
```

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 in the clear, 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 that with the public viewability of many chess players' email addresses from their profiles, and someone could quickly acquire a new Paypal account he need never fund.)

An active adversary can 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


ICC Help: timestamp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address s and Settings\jrblack\My Documents\papers\icc\ICC Help timestamp.htm
Go
Google
HOME JOIN ICC HELP MEMBERS & ACTIVITIES CHESS.FM RESOURCES STORE
ICC
THE INTERNET CHESS CLUB
Over 30,000 Members Worldwide
CHOOSE A LANGUAGE
QUICK LINKS
Currently online: 2550 players, including 25 Grandmasters and 50 International Masters.
HELP FILES
ICC Commands
ICC Information
HELP Top
http://www.chessclub.com/help/info-list
Internet

"If you are looking for professional management and strong competition, then the ICC is the best chess server." — Grandmaster Roland Schmaltz, The Complete Chess Server Guide (2004)

Get a seven day FREE trial of ICC!

ICC Help: timestamp

We have developed a system that eliminates the effects that lag has on your clock in ICC games. A "timestamp" program measures the amount of time you spend thinking about each move. The ICC server uses this information to update the clocks. You can ONLY be flagged if you have actually used more than your allotted time. You will NEVER be lag-flagged again!

Timestamp is built into the interfaces Blitzin, Fixation, IC for Mac, and WinBoard. It will run automatically any time you use one of these interfaces. You don't need to do anything special to use timestamp with these interfaces. You can download interfaces from our web page at "www.chessclub.com".

For other interfaces, you run timestamp in addition to your usual ICC interface program. It has been tested with xboard, xics, ziics, and giics, slics, MacICS, and other interfaces. There are two main versions of timestamp: Unix timestamp and MS Windows timestamp.

Unix timestamp works with any client if you connect to ICC through a Unix machine. Of course it works if your own machine runs Unix. It also works if your own machine does not run Unix, but you connect to ICC through a shell account on a Unix machine. In the latter case, though, timestamp cannot compensate for any lag that might exist between your own machine and the Unix machine where you run timestamp. Such lag can occur (for example) if the Unix machine is heavily loaded, so that you are not getting enough CPU time, or if you are calling over a noisy phone line with an error-correcting modem.

You can use MS Windows timestamp if you have a PC running Microsoft Windows that is on the Internet (most likely using SLIP or PPP), and your Internet software package supports the Winsock API. Nearly all do. MS Windows timestamp works with all the Windows clients, including SLICS, Raja.

All data sent in both directions between the timestamp program and the ICC server is encrypted. It can be used to send sensitive information (such as credit card numbers) over the network without worrying about eavesdroppers.

To find out how to get and use timestamp, type help unix-timestamp if you need the Unix version, or type help win-timestamp if you need the MS Windows version.

Figure 3: *The ICC timestamp help page explaining the benefits of timestamping. We have placed a box around the security claim near the bottom of the page.*

is similar to the DRM 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 Cheating Timestamp Client

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

Withholding the Source. Sleator 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. (For example, the source code for Microsoft Windows is tightly controlled.) 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.

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 Sleator run the Linux program `strip` in order to remove symbols from the binary before distributing it.

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]. In Sleator's defense, however, there were probably not very many obfuscators around in 1992 when timestamp was invented.

Hacking the Timestamp Program. Our first cheating timestamp program was a hacked version of the Linux program which simply zeroed the `eax` register, indicating that 0 seconds were used to make a move. (ICC charges a minimum of 0.1 seconds per move, so in fact one does not get an infinite clock with this technique.) Our method was to simply use `gdb` to overwrite instructions in-place to fill the `eax` register with 0. We overwrote a portion of code at the end of a function which normally repairs the call stack, and with a few more adjustments we repaired the stack later on. Finding the appropriate section of code to modify was easy because of the existence of symbols. Making this modification took about 30 minutes from start to finish.

Without symbols our job would still have not been too hard: the timestamp program must somehow determine the time, and this requires a call to the operating system which is easily identified. Modifying the code at this point would do the trick. Also, one could intercept the operating system call to `get_time_of_day` and return falsified values even if the timestamp program itself were inscrutable. In this sense, an obfuscator

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

would be of little help. Nonetheless, it would have taken a lot more than 30 minutes to achieve this, and there are a few additional technical problems to be overcome.

Our Cheating Client. The most general solution is, of course, to reverse engineer the client. Given its small size, this took us about 65 hours of work. Zeroing out a register, as we did in our first attempt, works fine but arouses suspicion: who, other than a chess program, responds instantly to every move? With our reverse-engineered client, more sophisticated (and therefore less suspicious) rules can be established for deducting time. 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.

Possible Remedies. Sleator undoubtedly knew that it was not too hard to make a cheating timestamp client. He was faced with trying to make Internet Chess a fairer experience without going to extraordinary lengths. To this end, he did about as well as could be expected.

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 be prohibitive, not to mention the endless problems with compatibility, diversity of platforms ICC supports, availability of bus slots or USB ports, and so forth. This solution is likely not practical.

A final suggestion that *is* practical would be for the ICC server to attempt to detect cheating timestamp clients out-of-band. The server could use various network services such as ICMP ECHOes (pings) to attempt to measure actual latency between itself and the client. If a timestamp process were sending back timings indicating lag far above the client's ping time, the server might choose to disable timestamping for that connection. Of course we could hack the client machine to send back inflated ICMP pings as well, but this begins to get much harder than the work we have done above. Additionally there are other network services which could be used in attempt to validate latency; hacking all of them would be quite an undertaking. Also, the server could ping upstream points from the client, deciding whether the lag was at the last-hop (suspicious) or somewhere along the way (less suspicious). It would be very difficult to hack backbone routers' ping times.

One promising approach might be to send a small program, perhaps as Java bytecode, which contains an authentication key embedded within it, and is obfuscated. This small program would be sent with each move, would measure the time used, and would output the time used along with an authentication tag. The program would be generated anew for each move, so a client would likely be unable to reverse engineer the obfuscated program quickly enough to be able to cheat. The problem with this approach is that the program would need to internally compute the elapsed time (since, as we have seen, calling the operating system is not secure). Still, this approach holds promise perhaps for other similar applications.

4 Cryptanalysis

In the previous section we showed how to simply defeat the timestamp security of ICC, and reached the somewhat unsatisfactory conclusion that there probably was no perfect solution to the problem. In this

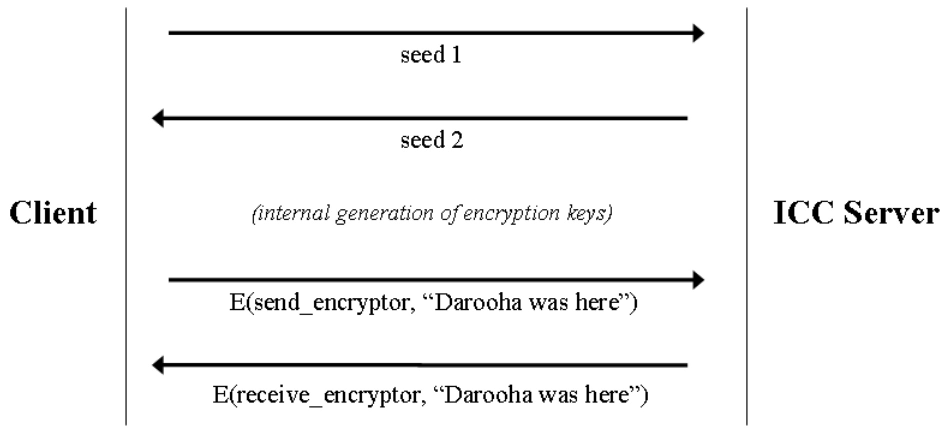

seed 1
seed 2
Client
(internal generation of encryption keys)
ICC Server
E(send_encryptor, "Darooha was here")
E(receive_encryptor, "Darooha was here")

Figure 4: *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 Sleator's ICC nickname.)*

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.

All algorithms above were (we suppose) invented by Sleator himself, as we have not seen these techniques used elsewhere in our experience. All of them have serious defects which we now discuss.

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. (As it turns out, the Linux client always chooses the *same* 64-bit string because the client fails to call `srand()` to seed the pseudo-random generator; the Blitzin client and the ICC server do not have this defect.) The server and client then exchange these seeds *in the clear*. 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 4). 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.

Our 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.)

Our sniffer also works with the Blitzin client, although there was additional work required here as the Blitzin protocol uses a deterministic transformation on the plaintext before encryption in order to compress the messages by about 30%. To extract this information we disassembled a portion of the Blitzin client, finding the relevant area of code in the Blitzin binary by searching for constants used in the encryption algorithms and then tracing execution in the neighborhood of the encryption calls. 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.

Once again, we do not plan to release our code.

Remedies. The simplest remedy would be to remove claims that ICC encryption provides any measure of security. Then require users wishing to pay online to submit membership fees through a web-based payment gateway using SSL/TLS. This would vastly increase the security of the system (in spite of the fact that payment gateways are periodically compromised). Note that the passive and active attacks already mentioned remain possible in the absence of a secure end-to-end protocol, but securing credit card information should probably be a top priority.

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 [21] could be used to quickly insert this functionality (at the cost of expanding the size of the client binary). Of course OpenSSL did not exist in 1992, so it is understandable that Sleator did not use it originally, but the secure key exchange techniques listed above were known for at least 15 years prior.

Once the server's RSA public key is present on the client, the client generates a pseudo-random session key (taking care that the pseudo-random generator uses proper techniques [8]) and encrypts using the server's public key. The server, having the corresponding private key, decrypts the session key and a symmetric algorithm is then used for further communication. This is all quite well-known and standard.

Implementing a full PKI system with key freshening, revocation, and expiration would probably be prohibitive. Perhaps an acceptable solution would be to fix a large (say, 4096 bit) RSA public key in the client and never change it. It should be some time before factoring technology and/or quantum computers represent a threat, and when they do we will have bigger problems than broken chess clients.

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 5). Specifically, $r_i = y_i \oplus z_i$ where

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

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

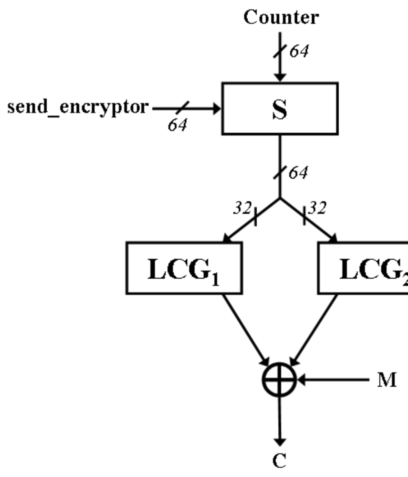


Figure 5: *The ICC Mode of Operation: 64 bits are generated by enciphering a counter under the sender's blockcipher key. The first 32 are sent to LCG$_1$ and the other 32 to LCG$_2$. The LCG's are iterated 100 times and their lowest bytes are XORed to produce an encryption pad which is XORed with the message M to produce the ciphertext C.*

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 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:

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

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

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 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).

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 $f : \{0,1\}^{64} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$ 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:

[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 $2/2^8$.

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.

REMEDIES. Once again the best remedy here is to throw out both the mode and the blockcipher and use something standard. Authentication is needed as well. Using a secure key-exchange protocol along with a provably-secure authenticated encryption scheme [3] would be the most efficient and cheapest solution. Alternatively, using a freely-available crypto library like OpenSSL [21] would provide a number of well-tested routines implementing standard protocols.

Of course even these approaches have their perils: implementation errors can lead to a complete loss of security. But finding these errors would surely be harder than exploiting the simple vulnerabilities uncovered in our analysis.

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 still have no fully-satisfying method for measuring the amount of time taken by the client. A solution to this problem would be useful in other domains beyond Internet Chess, though solving it cheaply and accurately seems to be difficult, analogous to the problems with DRM.

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[19] Sleator, D. D., and Tarjan, R. E. Self-adjusting binary search trees. *Journal of the ACM 32*, 3 (1985), 652–686.

[20] Stubblefield, A., Ioannidis, J., and Rubin, A. Using the Fluhrer, Mantin, and Shamir attack to break WEP, 2001. ATT Labs Technical Report, TD4ZCPZZ, available at `citeseer.nj.nec.com/article/stubblefield01using.html`.

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

A Legalities and the DMCA

Obviously this project involved a significant amount of reverse engineering, which remains under a legal cloud specifically with regard to the DMCA. We have done our best to ensure that our practices were in accordance with what is "allowable" for this type of research, and we have spoken with people at the EFF in hopes of fending off lawsuits against the authors and any venue which might wish to include our findings.