The following paper was originally published in the

Proceedings of the 8th USENIX Security Symposium

Washington, D.C., USA, August 23–26, 1999

BRUTE FORCE ATTACK ON UNIX PASSWORDS WITH SIMD COMPUTER

Gershon Kedem and Yuriko Ishihara

Brute Force Attack on UNIX Passwords with SIMD Computer

Gershon Kedem, Yuriko Ishihara
Computer Science Department
Duke University
Box 90129
Durham, NC 27708-0129

Abstract

As computer technology improves, the security of specific ciphers and one-way hash functions periodically must be reevaluated in light of new technological advances. In this paper we evaluate the security of the UNIX password scheme. We show that the UNIX password scheme is vulnerable to brute-force attack. Using PixelFlow, a SIMD parallel machine, we are able to "crack" a large fraction of passwords used in practice [12] in 2-3 days of computation. We explain how a SIMD machine built in today's technology could "crack" any UNIX password in two days. We also describe in this paper a simple modification to the UNIX password scheme that makes it harder to break encrypted passwords using dictionary and brute force attack, thus extending the useful life of the UNIX password scheme. The modified password scheme is compatible with the existing password scheme.

0. Introduction.

Single Instruction Multiple Data (SIMD) machines is a class of parallel machines that are made of a large number of simple processors all executing in unison the same instruction sequence, each processor computing on a different data set. It is relatively inexpensive to build SIMD machines with a very large number of processors. Since the processors are simple (typically 8-bit processors), processors use near neighbor connections, and processors do not use local instructions decoding, it is possible to integrate a large number of SIMD processors into a single IC. In current technology (0.2μ CMOS) it is possible to integrate 512-1024 processors each with 1-2 KB of RAM on a single IC. Moreover, since instruction decoding is external to the processors, it is easy to pipeline the processors and make them run at high frequency. In today's technology it is possible to build a SIMD processor array that execute one instruction per clock cycle at 1-GHz.

SIMD machines are very effective for performing regular computation on large data sets. Examples of applications that require large computing power and could effectively use SIMD machines are: real-time image analysis and real-time image generation. Both applications do relatively simple calculations on a large set of pixel data. Computation at each pixel is, for the most part, independent from the values of all but a few neighboring pixels.

SIMD machines are currently out of vogue since SIMD machines perform poorly on applications that require complex decisions and applications that have complex data dependencies. In either case the SIMD processor array could not be used effectively. In the first case, only a small fraction of the processors actually do useful work. In the second case, the machine spends a large portion of the time moving data between processors rather than doing useful work. SIMD machines are notorious for being hard to program. The SIMD machine programming-model is very different from the conventional programming model, and it is hard to port applications that run on conventional machines to SIMD machines.

However, SIMD machines are very effective for brute force cryptanalysis. First, most ciphers are very simple algorithms, made up of loops and straight-line code. Therefore, it is relatively straightforward to implement ciphers on SIMD machines, and the resulting programs harness the full power of the machine. Second, brute force cryptanalysis is "embarrassingly parallel". By assigning a different key to each processor it is possible to simultaneously check a large number of keys. The processors do not communicate, and all the processors execute most of the time. As a result, brute force cryptanalysis can achieve a system performance that is close to the theoretical performance-limit of the machine.

We have used the PixelFlow machine [2] to show that UNIX passwords are vulnerable to brute force cryptanalysis. The PixelFlow machine is an experimental graphics engine built at the Computer Science Department at UNC-Chapel Hill in cooperation with Hewlett Packard Corporation. The PixelFlow machine includes a large SIMD array of 8-bit pixel processors running at 100MHz. We programmed 147,456 PixelFlow SIMD processors to do brute force cryptanalysis of 40-bit RC4 cipher and the UNIX `crypt` password scheme.

In section 1. we describe the PixelFlow machine, and the SIMD array we used for the computation. Section 2. describes a brute-force attack using PixelFlow and its implications for UNIX security. We also describe the capabilities of a SIMD machine that one could build with today's technology and its security implications.

In section 3. we propose a way to modify the UNIX password scheme to make it resistant to brute force crypt-analysis.

1. The PixelFlow Machine.

PixelFlow is a heterogeneous parallel machine designed for a special purpose, high-speed and high-quality image generation. A PixelFlow system consists of at least one chassis, which includes up to nine Flow units. Each Flow unit has a SIMD array of 8,192 (8K) Processing Elements (PEs) running at 100MHz.

As shown in Figure-1, each Flow unit includes both a Geometry Processor (GP) board and a Rasterizer Board (RB). The GP board has two 180 MHz. PA-RISC-8000 CPUs, 128MB SDRAM memory, and a custom ASIC, the RHInO (Runway Host and I/O) that connects the processors with memory, the geometry network and the Rasterizer Board. The geometry network is a high-speed packet-routing network that connects the GPs to each other. In addition an I/O interface card connects the network to a host workstation.

The heart of the Rasterizer board is a SIMD (128x64) array of 8192 processing elements (PEs). The PE array is implemented on 32 logic-enhanced memory chips (EMCs), each containing 256 PEs. Figure-2 shows a block diagram of an EMC. Each PE is an eight-bit wide processor consisting of an arithmetic-logic unit (ALU), two registers and 384 bytes of local memory. This includes 256 bytes of main memory and four 32-byte partitions associated with two I/O ports. A linear expression evaluator computes values of the bilinear expression $A \bullet x + B \bullet y + C$ in parallel for every PE; the pair (x,y) is the address of each PA in the SIMD array. In addition, the Rasterizer board has a Texture/Video Subsystem that we did not use and will not describe here. For a full description of the PixelFlow system the reader should refer to [1, 2, 3].

Two Image Generation Controller ASICs (IGCs) control the rasterizer. The IGCs parse the instruction stream from the Geometry Processor board and issue micro-instructions to the SIMD PE array. A DMA unit transfers a stream of instruction from the GP SDRAM memory to IGC units and the SIMD array.

To program the SIMD array one loads a program into one of the GP CPUs. The GP processor, by executing the program, generates a sequence of microcode instructions for the SIMD array. The GP is used to control the SIMD array and to communicate with a host workstation. Since the system was designed for image generation, the SIMD array has a high bandwidth connection to the Texture/Video subsystem, but only a very limited (1-bit) connection back to the GP.

The programming environment for the SIMD array is limited. A set of **C++** functions implements an assembly language instruction set for the SIMD array. A functional level simulator is used for program development and debugging. The SIMD-array instruction set was designed to efficiently execute image generation code. Instructions can operate on a single byte, two, four, or eight bytes of data. Most instructions take two or three clock cycles per byte to execute. The SIMD instruction set is quite conventional except the set of linear expression-evaluation instructions. The instructions are memory to memory instructions. The instruction set includes the usual integer and bit-wise logical

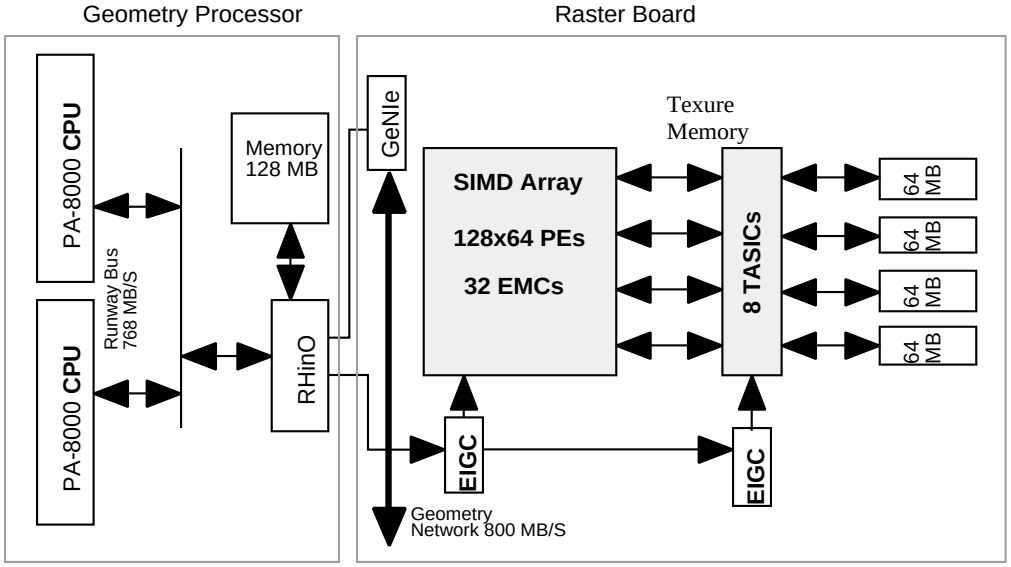

Figure-1: PixelFlow Unit, Block Diagram.

instructions. However, the PixelFlow SIMD array is missing indexing instructions. The PixelFlow hardware does not support the ability to take a memory value and use it as an index into memory. The PixelFlow machine was designed with image generation in mind, and the designers did not see the need for such capability. It turned out to be a major performance issue for implementing both the RC4 cipher and the UNIX `crypt` function (See section-2).

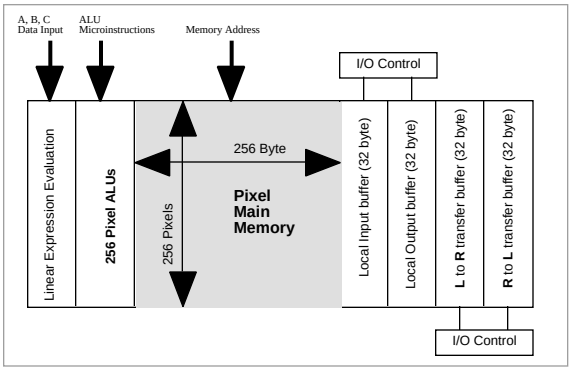

Figure-2: PixelFlow Enhanced Memory Chip (EMC), Block Diagram.

2. Brute Force Cryptanalysis with PixelFlow

Brute force cryptanalysis is the most straightforward cryptanalysis attack. The attacker uses raw computing power to find a key for the cipher by trying all possible keys. Typically the attacker has either a ciphertext-cleartext pair or ciphertext only. When the attacker is analyzing passwords, only the encrypted password is needed. Typically, brute force attack is carried either by using a collection of general-purpose computers or by using special-purpose hardware.

General-purpose machines provide a very flexible cryptanalysis platform. The machines can be programmed to attack many different ciphers. Nowadays when most computers are networked, it is not unusual to mount a brute force cryptanalysis attack using hundreds or thousands of computers. The attack is carried most often at off-hours, using the computers' "idle cycles". However, general-purpose computers are relatively slow. Using today's general-purpose computer cluster, it is practical to attack 40-bit keys, but 56-bit or 64-bit ciphers, in practice, are still out of reach.

Building special-purpose hardware for brute force cryptanalysis provides a very effective (but expensive) way to attack ciphers. Using today's technology (say, 0.2μ CMOS technology) one can build special-purpose machines that can "crack" 56-bit and 64-bit ciphers in a few seconds to a few days of computation. However, building special purpose hardware is time consuming and very expensive. A typical hardware development project could cost $10+ million and could last two to three years. At the end, all one has is a machine that can break a single cipher. If breaking the cipher is important enough, this approach makes sense; otherwise special-purpose hardware is too expensive for all but a few very large organizations or governments. Cryptanalysis attack using special-purpose hardware could be defeated by choosing another cipher or by making slight modifications to existing ciphers.

Building a programmable machine for cryptanalysis is an in-between approach. On one hand, the machine is programmable and could be used to analyze different ciphers; on the other hand, the machine could bring large computing power to the attack. Blaze et. al. [13] proposed using Field Programmable Gate Arrays (FPGAs) for mounting such an attack. Using FPGAs to implement ciphers in hardware could still require a large design effort. Designing circuits to implement ciphers at gate-level is a hard and error-prone task. Debugging the system is also hard and error-prone. However, once the design is complete, it could be mapped relatively quickly into FPGA arrays. FPGAs suffer from two problems: capacity and speed. FPGAs devote large portion of the silicon area to switches and wires. As a result, the number of gates one could actually use is much lower than the number of gates the technology supports. Second, since the on-chip gates are connected via switches, in practice the system clock speed is much lower than advertised. As a result, using FPGAs one could only built cryptanalysis systems with moderate performance.

We set out to evaluate the use of SIMD machines for brute force cryptanalysis. As was explained in the introduction, SIMD machines could provide cost effective way to mount brute force attacks. SIMD machines are fully programmable, and therefore it is quite easy to implement complex algorithms and search strategies. As noted before, when running brute force cryptanalysis code, the machine can approach its theoretical performance limit.

We used PixelFlow, an existing SIMD machine, in order to answer the following questions:

a. How hard is it to program ciphers and complex search strategies on a SIMD parallel machine?
b. What performance can one get out of the system when mounting realistic attacks?

c. What special features should SIMD systems have so they could support effective cryptanalysis attacks?
d. How powerful is the PixelFlow system? Can we compromise the security of ciphers used in practice?
e. If one is to design a new SIMD machine using current technology (0.2μ CMOS), a machine designed specifically for brute force crypt-analysis, how powerful a system one can build and at what cost?

We programmed two ciphers for the PixelFlow machines: RC4 [11] and the UNIX `crypt` algorithm [5, 6, 8, 9, 11]. The programming was done in machine assembler. Learning how to use the tools took longer than we expected, but the programming task proved to be no harder than writing assembler for any machine. The only difficulty we ran into was the fact that PixelFlow does not have index registers and it does not gracefully support table lookups. This turned out to be a major performance issue. On a "normal" machine array-lookup takes one or two instructions. On PixelFlow it takes time proportional to the array size. Using some clever programming, we were able to reduce it to time proportional to the square root of the array size [4]. Since the RC4 and DES algorithms both are dominated by table lookups, the performance we were able to achieve fell far short of the performance we expected. We estimate that a PixelFlow machine that supports indexing would run the RC4 code ~32X faster and run DES ~16X faster.

We used a PixelFlow system with two full chassis for a total of 18 SIMD arrays. Each array has 8K (8192) PEs, so the total number of PEs we used is 147456. At any given iteration the machine checks 147456 different keys in parallel. We used the linear expression evaluation unit to distribute a unique key to each PE. All PEs run in unison, and at the end of each iteration, the PEs check to see if any of the keys produced a match. The RC4 algorithm was timed at approximately 1000 key checks every 3.725 seconds per each PE. Since we have 147456 PEs, the system checks 38,804,210 keys per second. This enables us to check all 40-bit combinations in ~28,335 seconds (~ 7.87 hours) and on the average find a key in less than four hours.

2.1 Checking UNIX Passwords.

Checking UNIX passwords proved a bit more complex. To be able to check passwords rather than DES keys we devised a simple algorithm that translates a number into a unique password in a given class. Assume that we want to check all possible passwords with lower case letters. We construct a character array with 26 characters with the lower case letters. Translating the number *I* into the password *P* is just a base translation from binary to base 26, using the characters as the base digits. This assigns a unique password to each number. Simple modifications to the base translation algorithm let us check different classes of passwords. For example, checking all passwords with lower case letters and one digit in the third position is done with a base-change to base-26 in all positions except a base-10 change in the third position. Similar modifications let us check many different password classes. The idea is to check password classes that are most commonly used in practice. Eugene H. Spafford reports [12] that 28.9% of all passwords observed in his study had lower case letters only, and 38.1% had a mixture of lower and upper case letters. Brute-force attacks on these two password-classes alone make it very likely that a password would be compromised in a short time. For example, Table-1 shows that all lower case only passwords could be checked in 3.19 hours.

The UNIX `crypt` algorithm took approximately 6 seconds per 1000 passwords checked per PE. This yield a system level performance of 24,576,000 UNIX passwords checked per second (or 614,400,000 DES encryptions per second). Table-1 shows the time it would take to check all passwords in a given class. In the table we used LC for lower case letters, UC for upper case letters, D for digits, and P for all other characters.

2.3 Building SIMD machine in today's technology

The PixelFlow machine was built in the mid-90's using 0.5μ CMOS technology with three layers of metal. Today's CMOS technology has 0.18μ-0.2μ features, and the technology supports up to six layers of metal. Linear shrink alone yields a factor of ~6x improvement in density and a factor of ~4x improvement in speed. The PixelFlow machine was designed for high-speed image generation, and the SIMD array is only a small part of the overall system. Moreover, the SIMD array was not optimized for performance since it is already more powerful than other system components. The SIMD array is never the limiting factor in image generation, and rarely if ever is it used to its full potential. If one is to design a SIMD machine for brute force cryptanalysis, one could do better than just scale up the PixelFlow design.

We have constructed a "paper design" of a SIMD machine optimized for brute force cryptanalysis, using what we have learned from programming PixelFlow. Most of the design decisions for PixelFlow were carried over to our new design. The system is a heterogeneous machine made up of many SIMD arrays. Each SIMD array is controlled by a general purpose (GP) computer that also serves as a communication controller. The

SIMD arrays are connected to an ethernet network and are controlled by a general-purpose workstation. Like PixelFlow the GP processor has large SDRAM memory. The GPs are used to generate instruction streams for the SIMD array, and a DMA controller is used to load the instruction into the array.

The processing elements (PEs) are upgraded to improve on the PixelFlow design. Like in PixelFlow, processors are 8-bit wide. Each processor has 1K bytes of local memory and a set of 32 registers. The processors use a four stage pipeline (register fetch, execute, memory access, register store). Pipelining should improve system speed by 3x-4x. We (conservatively) estimate that 512 processors could be integrated into a single IC. We estimate the clock speed to be 1 GHz. Each SIMD array has 64 processors' ICs (32K PEs) with its own GP, SDRAM buffer and ethernet connection. Each SIMD array is implemented on a single printed circuit board. We estimate the replication cost of each SIMD array board to be $3,000. For a replication cost of about $100,000 one could build a system with 1,048,576 (1Meg) PEs. We estimate that such a system will deliver at least 1000X better performance than the PixelFlow system we used. Table-1 also lists the estimated time it would take to check different password combinations on the new machine.

Class	Combinations	PxFl hours*	New Design hours*
LC only	2.82E+11	3.19	0.003
LC + 1-UC	2.18E+12	24.59	0.025
LC + 2-UC	1.47E+13	165.77	0.166
LC + 1-D	8.37E+11	9.46	0.009
LC + D	3.51E+12	39.70	0.040
LC + UC	6.23E+13	703.71	0.704
LC + UC + 1-D	9.40E+13	1062.20	1.062
LC + UC + D	2.18E+14	2467.86	2.468
All passwords	5.13E+15	58008.14	58.008
All DES keys†	7.21E+16	32578.12	32.578

Table-1: Times for checking different password classes. (*Estimated, †DES only)

3. Improving Password security.

Our brute force attack experiment, using PixelFlow's SIMD processor arrays, demonstrates that UNIX passwords are vulnerable to such an attack. Moreover, we argue that it is possible to build a SIMD machine that could check all possible UNIX passwords in about two days. Some UNIX installations require that users use "safer" passwords. They instrument the UNIX `passwd` program to force users' passwords to have characters from at least two or three of the four categories: upper-case letters, lower-case letters, digits, and other-characters. This modification to the UNIX `passwd` program only marginally improves performance against brute force attack, since the attacker can use that knowledge to direct her/his attack. For example: there are 1.67E+12 passwords that have one upper-case letter and the rest are lower-case letters. The "more complex" class of passwords that have exactly one upper-case letter, one digit and the rest lower case letters has 4.50E+12 passwords, only a factor of 4 more passwords. This factor is not large enough to make a difference. We also argue that it is possible to build machines that are able to "crack" any UNIX password in a day. Therefore we are recommending that the UNIX community adopt a modification to the existing password scheme. This modification is backward compatible; it is tunable to the hardware "state of the art," making it possible to make the password scheme more secure when common computers become faster, while keeping the login time to about a second.

The main idea is to add random bits to the password encryption, similar to the "salt" bits that are currently used to protect passwords against dictionary attacks. We call the new bits: "pepper" bits. Unlike the "salt" bits that are saved with the encrypted password, the pepper bits are used to encrypt the password, but are never saved.

Let say that we are using k pepper bits for password encryption. The k-bit vector P is repeated as many times as necessary to form a 64-bit word V (V=<P, P, ••• >). Let E(***e***, ***s***, **0**) stands for the normal encrypted password, where ***e*** are the password bits, the ***s*** are the salt bits, and **0** is a 64-bit zero constant that is encrypted to get an encrypted UNIX password. Then the new encrypted password is: V+E(***e***, ***s***, V), where + stands for bit-wise exclusive-or. That is, the `crypt` algorithm is applied to the 64-bit word V (rather than to **0**) and the result is XORed with V. Note that if the pepper bits are all zero, V is zero and the new scheme is identical to the current scheme. When a user selects a new password the system generates a random k-bit vector P. The system generates the vector V by repeating P as many times as necessary, and it uses <e, s, V> to generate the encrypted password. The system saves the encrypted password together with the salt bits, but it *does not save* the vector V. When a user logs in, the system checks the password supplied by the user as before. The system runs `crypt` 2^k times with all the

possible values for V, computing V+E($\boldsymbol{e}$, $\boldsymbol{s}$, V). If any of the 2^k possible values match the encrypted password, the login succeeds. Otherwise the login fails.

What we propose here is essentially a "whitening" technique [11]. Our proposed UNIX password scheme is similar to a proposal made by Udi Manber [7], but it improves on Manber's scheme. Unlike the scheme proposed by Manber, if one chooses k carefully, the probability of hitting a false positive <password, pepper> combination is smaller than the one proposed by Manber. The probability that V_1+E($\boldsymbol{e}$1, $\boldsymbol{s}$, V_1) = V_2+E($\boldsymbol{e}$2, $\boldsymbol{s}$, V_2) when $V_1 \neq V_2$ and $\boldsymbol{e}1 \neq \boldsymbol{e}2$ is $2^{-(64-k)}$ since the encryption function is a strong one-way hash function, both as a function of the ciphertext and as a function of the keys. If one chooses k=11, the probability of hitting a false positive combination is 2^{-53}, about the same probability as guessing a random password. Our scheme protects passwords against dictionary and brute force attacks by forcing the adversary to do 2048X more work, while keeping the probability of "false positive" guesses to the same probability of guessing a random password. Our experiments show that on current machines, adding 11 pepper bits keeps login time to about a second.

4. Conclusions:

In this paper we study the use of SIMD machines for brute force cryptanalysis. We show that SIMD parallel machines are very effective. We demonstrate that an existing machine, PixelFlow, can break many UNIX passwords that are used in practice. We argue that it is possible and practical today to build a machine that could "crack" any UNIX password in a day or two. Moreover, this machine could be programmed to break ***any 56-bit cipher*** in two days. We proposed a simple modification to the UNIX password scheme that makes the scheme 2048X more resistant to dictionary and brute force attack. We argue that as time goes by and computer hardware becomes faster and less expensive, reusable password schemes are becoming more vulnerable. The source of vulnerability is the fact that people must remember the password and should not keep a written copy of the password or store the passwords electronically. In practice, the set of passwords that people can remember is too small to offer strong protection against adversaries with large computing resources. The community will be well served by introducing new authentication schemes.

Acknowledgements:

PixelFlow was designed and built by a research team at the Department of Computer Science at UNC-Chapel Hill and the Hewlett-Packard Corporation. We are grateful to the UNC team for helping us use PixelFlow. Special thanks are due to Anselmo Lastra and John Eyles for their help.

References:

[1] Lastra, Anselmo, Steven Molnar, Marc Olano, and Yulan Wang, "Real-Time Programmable Shading", *Proc. of the1995 Symposium on Interactive 3D Graphics, ACM Siggraph*, April 1995.

[2] Eyles, John, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, and Nick England, "PixelFlow: The Realization", *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, Los Angeles, CA, August 3-4, 1997, 57-68.

[3] Eyles, J. and Molnar, S., "PixelFlow™ Raterizer, Functional Description". UNC CS Department internal document, 1997.

[4] Eyles, John, "Privet Communication". 1998

[5] Feldmeier D.C., and P. R. Karen, "UNIX password security – ten years later", *Proceedings, UNIX Security Workshop*, August 1989.

[6] Simon Garfinkel and Gene Spafford, *Practical Unix Security*, O'Reilly & Associates, Inc., Sebastapol, CA, 1991.

[7] U. Manber, "A Simple Scheme to Make Passwords Based on One-Way Functions Much Harder to Crack," Computers & Security 15 (2) (1996), pp. 171-176.

[8] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1996.

[9] Robert Morris and Ken Thompson, "Password Security: a case history". In *UNIX Programmer's Supplementary Documentation, AT&T*, November 1979.

[10] Poulton, J., J. Eyles, and S. Molnar, "Breaking the Frame-Buffer Bottleneck with Logic-Enhanced Memories," *IEEE Computer Graphics and Applications*, November 1992, pp. 65-74.

[11] Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, Inc. New York, 1996.

[12] Eugene H. Spafford, "Observing Reusable Password Choices", *In Proceedings of the 3rd Security Symposium*, Usenix, September 1992.

[13] M. Blaze, W. Diffie, R.L. Rivest, B. Scheier, T. Shimomura, E. Thompson, M. Weiner, "Minimal Key Length for Symmetric Ciphers to Provide Adequate Commercial Security", *A Report by an Ad Hoc Group of Cryptographers and Computer Scientists*. URL: http://theory.lcs.mit.edu/~rivet/bsa-final-report.ps