TDDC03 Projects, Spring 2006

Evaluation of Common Key and Password Generation

Hasham Ud Din Qazi Alexei Bavelski
Linköpings universitetet, Sweden

Email: {hasud353,aleba691}@student.liu.se

Viiveke Fåk

Evaluation of Common Key and Password Generation

Hasham Ud Din Qazi Alexei Bavelski
Linköpings universitetet, Sweden
Email: {hasud353,aleba691}@student.liu.se

Abstract

Today passwords are the most common mechanism to authenticate users, even for high security tasks like acknowledging financial transactions in companies (e-invoice handling, logging in as a financial trader etc). Most "secure" websites in e-commerce require their users to register via a self-chosen password. Also actual crypto keys are sometimes created from user entered passwords, as in Kerberos, PGP etc. In other cases cryptographic keys are generated by better routines, but still from fairly little randomness. This document analyzes vulnerabilities associated with passwords and related attacks. Also, the report touches the issue regarding randomness required for cryptography algorithms and its co relation with widely used authentication based systems.

1. Introduction

A password is a form of authentication data, which is kept secret and is used to moderate access to resources. This authentication paradigm is based on what people know i.e. a user provides a password to the authenticator and on the basis of provided information the system validates the password. Password information is kept secret from those who are not allowed to access the system.

Controlling access through passwords has spanned a lot with time. It is used in ATMs and mobile phones, in computer systems and TV decoders. Typically passwords are require for logging into the computer system, being part of a network, retrieving email from servers, accessing files, databases, e-commerce based websites, forums etc.

In some cases passwords are chosen by users, in other cases they are generated randomly by computer. Both methods have their pros and cons related to security issues. In case of user selected passwords, they might be too simple, repeat at many places, or be easily vulnerable to dictionary attacks. In case of computer generated passwords other problems appear. First of all it gets much harder to remember passwords and people start to write them in different places, where they could be found by potential attacker. Secondly, how random are computer generated passwords? Does the attacker needs to search all possible combination to guess the correct password or additional information might help her to reduce the search space.

In cryptography, key is some information that controls operation of a cryptographic algorithm. As in the case of passwords, key needs to be kept secret and only be available to authorized persons. According to Kerckhoffs' law, system should be secure, even if all details of the system, except the key are available to the attacker. As a result, security of the cryptographic system depends on the strength of the key. As in case of passwords, key size is the measure of the number of possible keys and key space represents the set of all possible keys. Brute force attack is analogue to dictionary attack in case of passwords and is searching of the key, running through the entire key space. In different systems key are generated in different ways. The security of the key has a strong relation with randomness. We study pseudo-random sequence generation algorithms in section 4 and different sources of randomness in section 5. Also, many systems use user entered passwords to generate cryptographic keys, that is applying various cryptographic functions, usually one way hash of passwords. Systems such as Kerberos, PGP, etc generate keys in the same fashion and are discussed in section 6. In this case attack on the key, basically becomes attack on the password. We study user chosen passwords in section 2 and randomly generated passwords in section 3. A password is a form of authentication data, which is kept secret and is used to moderate access to a resource(s).

2. User chosen passwords

A password usually is some information that confirms a user's identity. In the simplest case, it is a sequence of characters. The password space represents the set of all sequences of characters that are eligible to be passwords. Usually passwords are not kept in the system in clear form, but are mostly hashed using one-way hash function, resulting into a compliment. Resulted string is stored in a file or in a database. The goal of the authentication system is to ensure that users are correctly identified. If one could guess another user's password than one user could impersonate another user and the system's security would be compromised.

If A is a password space, the user password is a, such that $a \in A$. Using function f, such that $f \in F$, password a is transformed to a compliment string c such that, $c \in C$. If we take for example student cards PIN, it contains 4 digits, so its password space A consist of 10 000 elements (from "0000" to "9999"). In UNIX systems hashing functions f are based on permutations of Data Encryption Standard. All user passwords are hashed to 11-character strings and are stored in

separate file(s). Initially, back in time this file was accessible to every user, and could be easily stolen / cracked by the attacker. So, the security of the system was based on the criteria that the function could only do one-way transformations and if the attacker has saved the hash of the password *c* and even know-how of the transformation function *f*, he / she will not be able to obtain the password.
But the attacker can try to guess the password, calculates its hash, using the same function *f* and compare the hash. This is the simplest attack against the password-based system. In the worst case attacker will have to try all passwords in the password space.
So, one of the security factors is concerned about large password space. Assuming a password space of *120* characters, there are about *43,359,498,756,302,520*(*4.34 *10 ^16*) possible passwords of length one through eight. At *50,000* attempts per second, an exhaustive search of this password space would require over *27,480* years to complete.
But let's consider that how users choose passwords. For most of the humans it would be difficult to remember meaningless sequence of lowercase and uppercase letters, numbers, and punctuation marks. That's why they usually choose passwords that remind them of something. For example, someone's name, date of birth, telephone numbers, daughter's name, etc.
So here there is another threat from a specific type of attack: known as Dictionary attack. M. Bishop in his book "Introduction to Computer Security" [5] defines two types of dictionary attacks:

Type1:

Complimentary information and complementation functions are available and if *g* is the guess, if *f (g)* corresponds to complimentary information of the correspondent entry, then *g* is a correct password.

Type2:

either complimentary information or complementation function is unavailable and authentication function *l* may be used. In this case for guess *g*, if *l (g)* returns true, *g* is the correct password.
So attacker doesn't need to try all possible passwords from the password space, but rather use a dictionary.
In the article "Foiling the Cracker" [1], Daniel V. Klein describes a research made on user-chosen passwords. In his experiment he obtained a database of about 15, 000 account entries and tested each of the account entries by the number of intrusion strategies.

1. He tried using user's name, initials, account name and other relevant personal information as possible passwords (totally about 130 different passwords for this type).

2. Words from different dictionaries, totally about 60,000 separate words for each user. They consisted of names, keyboard patterns, numbers, common phrases, collection of words from different technical papers, etc.

3. Various permutation of words from step 2 (making the first letter uppercase, reversing the word, making the word in plural, adding different suffixes, etc.). These made 14-17 more tests per word, which are about 1,000,000 more possible passwords per user.

4. Tried foreign language words on foreign user names.

5. Tried word pairs.

As a result in about 12 months of CPU time, 25% of the passwords were recovered, 21% of the passwords were guessed in first week and 2.7% (368 passwords) were guessed in first 15 minutes of testing. In average, Daniel Klein argues that system with about 50 accounts could expect the first account to be cracked in less than 2 minutes, and 5-15 accounts cracked by the end of the day.
Interesting statistics of passwords, cracked in the Klein's research is represented in table 1.

Passwords cracked from a sample set of 13,797 accounts

Type of Password	Size of Dictionary	Duplicates Eliminated	Search Size	# of Matches	Pct. of Total	Cost/Benefit Ratio*
User/account name	130†	–	130	368	2.7%	2.830
Character sequences	866	0	866	22	0.2%	0.025
Numbers	450	23	427	9	0.1%	0.021
Chinese	398	6	392	56	0.4%‡	0.143
Place names	665	37	628	82	0.6%	0.131
Common names	2268	29	2239	548	4.0%	0.245
Female names	4955	675	4280	161	1.2%	0.038
Male names	3901	1035	2866	140	1.0%	0.049
Uncommon names	5559	604	4955	130	0.9%	0.026
Myths & legends	1357	111	1246	66	0.5%	0.053
Shakespearean	650	177	473	11	0.1%	0.023
Sports terms	247	9	238	32	0.2%	0.134
Science fiction	772	81	691	59	0.4%	0.085
Movies and actors	118	19	99	12	0.1%	0.121
Cartoons	133	41	92	9	0.1%	0.098
Famous people	509	219	290	55	0.4%	0.190
Phrases and patterns	998	65	933	253	1.8%	0.271
Surnames	160	127	33	9	0.1%	0.273
Biology	59	1	58	1	0.0%	0.017
/usr/dict/words	24474	4791	19683	1027	7.4%	0.052
Machine names	12983	3965	9018	132	1.0%	0.015
Mnemonics	14	0	14	2	0.0%	0.143
King James bible	13062	5537	7525	83	0.6%	0.011
Miscellaneous words	8146	4934	3212	54	0.4%	0.017
Yiddish words	69	13	56	0	0.0%	0.000
Asteroids	3459	1052	2407	19	0.1%	0.007
Total	86280	23553	62727	**3340**	**24.2%**	0.053

Table 1.

The total size of the dictionary was 62, 727 words.
An interesting research on user-chosen passwords was done in Purdue University. In the article "Observing Reusable Password Choices" [3] Eugene Spafford describes design of *password collector*, which was installed on the several systems for almost a year and collected passwords were studied for vulnerabilities to dictionary attacks. Total of 13787 unique password entries were examined. The average password length found to be 6.8 characters. Table 2 represents distribution of length, and table 3, distribution of characters.

Length	Quantity
1	55
2	87
3	212
4	449
5	1260
6	3035
7	2917
8	5772

Table 2.

Characters	Count	Percentage
Lower-case only	3988	28.9%
Mixed case	5259	38.1%
Some upper-case	5641	40.9%
Digits	4372	31.7%
Meta-characters	24	0.2%
Control characters	188	1.4%
Space and/or tab	566	4.1%
. , ;	837	6.1%
- _ = +	222	1.6%
! # $ % ^ & * ( )	654	4.7%
Other non-alphanumeric	229	1.7%

Table 3.

First passwords were compared to the information provided by users in the registration form. It was user name, phone number, and account name. 12839 unique words were derived from this information and 592 passwords (3.9%) were found to match. Second comparison was made against standard dictionary of the system (in their case SunOS 4.1.1) which consist 25144 words, 620 words were found to match. Next comparison was made with set of dictionaries in 11 languages, results are presented in table 4. Finally collected passwords were compared against large "miscellaneous" list of words from various collections and 2498 matches were found in this comparison.

Totally 2754 (20%) of the collected passwords were quickly found in different dictionaries and wordlists. Author also supposes that another 10% could match if perform some sort of simple transformation on the words from the dictionary.

According to the efforts represented above, we can argue that situation with user chosen passwords is not very good. Obtaining password file, attacker could recover at least several passwords and gain access to the system. Several solutions have been developed to the problem. One, most

Dictionary	Matches
Australian/Aboriginal	133
Danish	389
Dutch	313
English	1798
Finnish	1068
French	353
German	392
Italian	1087
Japanese	626
Norwegian	368
Swedish	261

Table 4.

obvious of course is hiding the password file from regular users. UNIX system uses *shadowing passwords*, and makes password file readable only by root. This is of course useful safeguard, but it doesn't make the system absolutely safe. Attackers could find different ways to obtain password file, which are not the topic of this report. Another approach, which is useful in addition to shadowing password file, is to eliminate easy-to-guess passwords. Some systems periodically run *password checker*, which scans password file and tries to break passwords using different dictionaries. When weak passwords are found, owners are advised or notified to change them. Main drawback of this approach is that it is quite time-consuming and attacker could always obtain more sophisticated or vast dictionary and is able to find more vulnerable passwords. Another approach is to force users periodically to change passwords. The biggest disadvantage here is that it doesn't helps the user to choose more complex passwords, in other way, its like implying the user to choose password from the same dictionary domain or use the same account information or the same pattern, which doesn't helps out.

Another solution to the problem is a *proactive password checker*, which would test user passwords before they are saved in the system. Daniel Klein [1] describes 14 points, which were detected in his research, and which could be used to create a proactive password checker:

- "Passwords based on the user's account.
- Passwords which exactly match a word in a dictionary.
- Passwords which match a reversed in the dictionary.
- Passwords which match a word in a dictionary with an arbitrary letter turned into a control character.

- Passwords which are simple conjugations of a dictionary word (i.e., plurals, adding "ing" or "ed" to the end of the word, etc.)
- Passwords which are shorter than a specific length.
- Passwords which do not contain mixed upper and lower case, or mixed letters and numbers, or mixed letters and punctuation.
- Passwords based on the user's initials name or given name.
- Passwords which match a reversed word in the dictionary with some or all letters capitalized.
- Passwords which match a dictionary word with the numbers '0', '1', '2', and '5' substituted for the letters 'o', 'l'.
- Passwords which are patterns from the keyboard (i.e., ''aaaaaa'' or ''qwerty'')
- Passwords which consist solely of numeric characters (i.e., Social Security numbers, telephone numbers, house addresses or office numbers).
- Passwords which look like a state issued license plate number".

Of course, password checker should be equipped with all necessary dictionaries and would be nice if in addition to the rejecting password, it could argue, why the password is rejected.

3. Randomly generated passwords

As it was mentioned in the last section that the main problem with user chosen passwords is that users usually choose their passwords that are weak in nature and they can be easily recovered (derived) by simple dictionary attacks. According to the theorem from the book by M. Bishop "Introduction to Computer Security" [5], we assume that time T expected to guess the password is maximum, when the selection of any of a set of possible passwords is equiprobable. Let try to evaluate time needed to guess password in this case [5]. Anderson's Formula shows that if S is the length of the password and A is the number of characters in the password such that $N = A^s$, G is the number of guesses that can be tested in one time unit, T – the number of time units, N – number of possible passwords, probability that an attacker guesses a password in time T is:

$$P \geq \frac{TG}{N}$$

For example, let say our passwords are chosen from the alphabet of 96 characters, and we assume that 10,000 passwords can be tested each second. Our goal is that probability of successful password guess to be 0.5 over a year period.

From the Anderson's Formula we get:

$$N \geq \frac{TG}{P} = \frac{(365*24*60*60)*10^4}{0.5} = 6{,}31*10^{11}$$

So we must choose S, such as

$$\sum_{i=0}^{S} 96^i \geq N = 6.31*10^{11}$$

In this case S should be at least 6.

Use of random passwords makes the system more secure, but a number of problems appear. One of them is that for users it is truly difficult to remember random-generated passwords because they don't remind them of anything and can't even be pronounced. A compromise between using random passwords, generated by the computer and user selected passwords, which are pronounceable in nature. Pronounceable passwords are based on the unit of sound called phoneme [5]. Phonemes that construct passwords are represented by the sequences of letters *cv, vc, cvc, vcv,* where *v* is a vowel letter and *c* a consonant letter. The idea is that pronouncing the password, user will not memorize single characters, but chunks of characters. Of course password space of pronounceable passwords is much smaller than of random passwords. Paper "A New Attack on Random Pronounceable Password Generators" [4] describes one of the possible vulnerabilities of pronounceable passwords. Because not all pronounceable passwords are easy to pronounce or remember, usually system generates several choices and let the user to choose one of them. The idea of the attack is that users choose passwords, which are situated in one subspace, more often than passwords in the other subspace. So the password space lacks the property of being equiprobable, the attacker can find the subspace and can deduce user passwords from this space and eventually will aid the user to only try passwords from this subspace.

4. Pseudo-random sequence generation algorithms

- **Blum Blum Shum Generator**

BBS (Blum, Blum and Shub generator) is one of the simplest and an efficient complex generator having a strongest public proof of its strength [6], BBS is also known as quadratic residue generator. BBS generators, in general are slow, in practise they are not useful for stream ciphers but there exist versions of this generator which speed it up, comprising on the "secure" aspect of it. These generators are used for public key cryptography [7].

The reason for its strong security and its lack of speed is that it generates pair of large sized prime numbers [8]. Also generation of large primes is much faster than

factoring the product of two of the said primes, so this makes the algorithm cryptographically more secure. As it is extremely slow compared to other RNGs [8], it is not appropriate for use in simulations, only for cryptography [9].
Experiments by L'Ecuyer and Proulx suggest that finding large special primes and an element in a long cycle may require 155 hours of CPU time (on a MicroVax II) for an improper 128-bit design [9].

Method:

The generator requires generating two large prime numbers, which are congruent to 3 modulo 4. The product of these integers is n also known as blum integer. Another random integer is chosen, x, which is relatively prime to n. Then x_0 is computed as following:

$$x_0 = (x^2) \pmod{n}$$

and follows as:

$$x_i = (x^2{}_{i-1}) \pmod{n}$$

Advantage:

The most efficient property of this generator is that you don't have to iterate through all i-1 bits to get the i'th bit. This means that in applications where many keys are generated in this fashion, it is not necessary to save them all. Each key can be effectively indexed and recovered from that small index and the initial *x* and *n*, if *p* and *q* are know the i'th bit can be computed directly. This property can use this cryptographically strong pseudo-random-bit generator as a stream cryptosystem for a random access file. BBS generator is unpredictable to the left and unpredictable to the right which means that the cryptanalyst cannot predict the next bit in the sequence or the previous bit in the sequence [10].
BBS is itself is cryptographically strong (or is believed to be), where the responsibility for providing a secure seed is left to the client using the PRNG (Pseudo Random Number Generator). BBS generator is likely to be used for high security applications, such as key generation (session keys) [11].

Analysis:
The security of this paradigm is dependent on difficulty of factoring n. The x^2 mod N RNG is claimed to be "unpredictable" (when properly designed), but even this is no absolute guarantee of secrecy. An attack on RNG repetition does not require "prediction." Even a brute force attack has the possibility of succeeding quickly. An inference attack could be practical if some way could be found to efficiently describe and select only those states which have a particular output bit-pattern from the results of previous such selections; that we currently know of no such procedure is not particularly comforting [12].

To put this in perspective, we should recall that all digital computer RNG's, including x^2 mod N, are deterministic within a finite state-space. Such mechanisms necessarily repeat eventually, and may well include many short or degenerate cycles. It is unnecessary to "predict" a sequence which will repeat soon. Accordingly, the x^2 mod N RNG requires some fairly-complex design procedures, which are apparently intended to assure long cycle operation [13].
If integer factorization is difficult (as is suspected) then BBS with large N will have an output free from any non-random patterns that can be discovered with any reasonable amount of calculation. This makes it as secure as other encryption technologies tied to the factorization problem, such as RSA encryption [13].

- **RC4**

RC4 is a stream cipher designed by Rivest for RSA Data Security (now RSA Security). It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. Independent analysts have scrutinized the algorithm and it is considered secure [14].
Its followers are RC5 and RC6 designed by Ronald Rivest for RSA Security, these ciphers use parameterized algorithms, such as, variable block size, a variable key size, and a variable number of rounds / iterations. Such built-in variability provides flexibility at all levels of security and efficiency.

Applications:

RC4 is used in various applications such as Lotus notes, Apple computer's AOCE and Oracle Secure SQL. RC4 is also used in wireless technologies such as WEP and WPA. Also MPPE (Microsoft Point-to-Point Encryption), SSL (Secure Sockets Layer) (optionally) and SSH (Secure Shell) (optionally).
RC4 is used for file encryption in products such as RSA SecurPC, RSA SecurPC is a software utility that encrypts disks and files on both desktop and laptop personal computers. SecurPC extends the WindowsTM File Manager or Explorer to include options for encrypting and decrypting individually selected files or files within selected folders [15].

Implementation:

RC4 generates a pseudorandom stream of bits (a "key stream") which, for encryption, is combined with the plaintext using XOR as with any Vernam cipher.

Many stream ciphers are based on linear feedback shift registers (LFSRs), and, while efficient in hardware, are much slower in software. The design of RC4 is quite different, and is ideal for software implementations, as it requires only byte-length manipulations. It uses 256 bytes of memory for the state array, S[0] through S[255], k bytes of memory for the key, key[0] through key[k-1], and integer variables, i, j, and k. Performing a modulus 256 can be done with a bitwise AND with 255 (or on some platforms, simple addition of bytes ignoring overflow) [16].

Analysis:

RSA Data Security, Inc (RSADSI) claims that the algorithm is immune to differential and linear cryptanalysis, have large cycles and is highly non linear. The algorithm is simple enough that most programmers can quickly code it from memory [10].
In 2001 a new and surprising discovery was made by Fluhrer, Mantin and Shamir: over all possible RC4 keys, the statistics for the first few bytes of output keystream are strongly non-random, leaking information about the key. If the long-term key and nonce are simply concatenated to generate the RC4 key, this long-term key can be discovered by analysing large number of messages encrypted with this key [17]. This and related effects were then used to break the WEP ("wired equivalent privacy") encryption used with 802.11 wireless networks. This caused a scramble for a standards-based replacement for WEP in the 802.11 market, and led to the IEEE 802.11i effort and WPA [18].

- **Linear Congruential Generator**

A Linear Congruential Number Generator (LCG) produces a sequence of numbers x_1, x_2, x_3 ..... where

$$x_n = a\ x_{n-1} + b \pmod{m}$$

In this relation x_0 is the initial seed, and values a, b and m are parameters that makes the relation for this generator. The generator has a period not greater than length m, b should be relative prime to m.

History:

The LCG is perhaps the most commonly used Random number generator in modern computer applications, but strangely, it was invented by D.H. Lehmer in a time when his concept had almost no practical use, as there were essentially no computers around [19]. It was only later on, when programmers required a fast way to generate a large stream of random numbers that Lehmer's LCG method was used as it simple and fast. These early programmers were more interested in speed than statistical randomness, and thus many of early LCGs were awfully flawed [20].

Most computers have a method for generating random numbers that is readily available to the user. For example, the standard C library contains a function rand () that generates pseudo-random numbers between 0 and 65535. This pseudo-random function takes a seed as input and produces bit stream. This rand () function and many other pseudo-random number generators are based on linear congruential generators [21]. Donald Knuth quotes "… look at the random number subroutine library of each computer installation in your organization… Try to avoid being too shocked at what you find [19]".

Analysis:

As cryptography requires values (bits) which are unpredictable. The use of pseudo-random number generators based on LCG is mostly suitable for experiment purposes or for simulations, and is highly discouraged for cryptographic functions because this generator is predictable even without knowing the value of the parameters involved in the relation. Eavesdropper can use the knowledge of some bits to predict the value of future bits with high probability. However they are efficient and show good statistical behaviour with respect to most reasonable empirical tests [21,10].
Regarding the ubiquity of flawed LCGs, this generator can be easily implemented and are fast, It is, however, well known that the properties of this class of generator are far from ideal. If higher quality random numbers are needed, and sufficient memory is available (~ 2 KBytes), then the Mersenne twister algorithm is a preferred choice [22].

5. Sources of Randomness

- **Cryptographic randomness from air turbulence in disk drives**

In article [23] it tells about cryptographic randomness from air turbulence in disk drives, as disk drive's motor speed variates irregularly due to the air turbulence present in the disk's closure. It shows by timing disk accesses, a program can extract about 100 independent, unbiased bits per minute, at no hardware cost.
Further on, the discussion motivates the idea behind generating random numbers from air turbulence in disk drives. As Secure Pseudo Random Number relies on the complexity of the algorithm, hardware providing natural physical noise is expensive also it tends to be biased and is correlated but idea of air turbulence is inexpensive, reliable and mathematically noisy.
On the other hand OS detects disk faults unlike other hardware (devices), so the randomness failure is unlikely. Secondly the disk can be secured from out side influence and observations, most importantly non linear dynamics

of disk aids to construct a mathematical argument on it. Turbulence occurs at the read/write heads and their support arms. The usage of Fast Fourier Transform acts as an unbiasing algorithm, converting variations form the access times into uniformly distributed and independent variable. The research provides proof of variance due to air turbulence according to various tests and work of others.

- **Randomness from Radio active nuclear decay**

Radioactive decay is the set of various process from which nuclides (atomic nuclei) emits radiation in form subatomic particles. Decay initiates in the parent nucleus and produces a sub daughter nucleus. This procedure is random in nature and is impossible to predict the decay of individual items. Geiger counter [24] can be used to detect alpha and beta radiation. This instrument amplifies input signals and displays it to the user.

On the premise that radioactive decay is truly random (rather than merely chaotic), it has been used in hardware random-number generators and is an invaluable tool in estimating the absolute ages of geological materials and young organic matter [25]. In [26] a true random generator, "RANDy", is proposed which is based on radioactive decay. They utilized the alpha decay of Americium 241, for generating random number generator for cryptographic applications. It shows three different algorithms for the extraction of random bits from the exponentially distributed impulses. Applying statistical test to their idea, confirming high quality of data delivered by the device.

The TRNG presented in this work consists of a radioactive source and a corresponding detector. The decay impulses are filtered and amplified for a further digital processing. The random bits are obtained by deciding whether the time interval between two pulses consists of an even or odd amount of timing units. This processing is done by a micro-controller that sends the random data via RS232 to a host computer where it is captured by a standard terminal program and stored or used. The length of the time intervals between two consecutive decay impulses is unpredictable. RANDy based on a preparation of the alpha radiator Americium 241 from a common household smoke detector so the total amount of radioactive material is not very critical. Geiger Muller tubes are rather common for simple qualitative measurements of mostly beta-radiation. On the other hand, a semiconductor sensor has the advantage that no high voltage is necessary and the recovery time after an impulse was detected is much smaller than in a tube. This method was applied to construct this TRNG.

- **Randomness Using Noise**

Natural noise can be used for TRNG (True random number generator) requiring special hardware. In [27] a random number generator using Wi-Fi background noise is discussed. It experiments with wireless technologies / devices (such as WLAN, Wireless LAN, or IEEE 802.11) recording background noise in their environment, relying on Electro magnetic noise as a physical phenomena. Today, information such as noise level can be queried from such applications. Command such as *iwconfig* can be used to extract the information (such as noise level) of a wireless device.

Furthermore [28] discusses the idea of using thermal noise with SHA 2 for generating equally uniformed random numbers, they retrieve thermal noise generated by integrated circuits. In addition to this, [29] shows the usage of metastability and thermal noise to generate random numbers demonstrating their results from a fabricated circuit, it specifies when a signal violates a bistable device's signal setup and hold timing requirements, the device output comes unstable. The undesirable phenomenon where the final state of the device is unpredictable and is determined by thermal noise is knows as metastability [29]. This study creates metastability from manufacturing variation in ICs.

- **Randomness using External Interrupts (System Clock)**

Working with randomness from natural sources is a big issue, it involves special hardware plus the process of extracting randomness is quite slow. Work has been done on using interrupts from external resources like keyboard strokes, hard-disk I/O completions, network packet arrivals etc [31] which result in uniformly distributed random sequence of 0's and 1's. This effort relies of not requiring a dedicated hardware and using the inherent entropy of external interrupts. Also lower bits of clock register when sampled on a lower rate, provides a good source of random bits.

The methods used a generator which is known as the entropy daemon and a buffer manager for management. Entropy daemon is responsible for collecting time stamps from external interrupts and converting them ultimately to a sequence of uniformly distributed bits. The buffer manager distributes the bits by serving blocking and non blocking read requests by processes in the operating system.

Like wise LINUX random number generator utilizes the same principle that is generating randomness from entropy of operating system events. The LINUX kernel uses the random data for various functions such as generating random identifiers, computing TCP sequence numbers, producing passwords, and generating SSL private keys. The interface of receiving random values from Linux Random Number (LRNG) Generator is the function *get_random_bytes(*buf, nbytes)*. The API to the LRNG is through two devices */dev/random* and */dev/urandom.*

/dev/random is used for more secure bits, while generating bits from this device the user might go in a halt state. */dev/urandom* is used for less secure bits [32].

- **User Input Latency**

It's been observed that people typing pattern is both random and non-random. The random pattern is random enough that they can be used to generate random bits. The process is carried by measuring time between successive keystrokes, and then extracting least significant bits of these observations. These bits may be biased and co related, but, can be distilled using whitening algorithms.
This scheme may not be suitable for UNIX terminal, as keystrokes pass through various transformations before they get to the program, but can be used in most of the personal computers[10].
Normally the extraction depends on reliance of data measured according per stroke. Normally one bit per type is suitable; extracting more bits might skew the result. This situation also depends on the typist that how good is a typist. The technique is limited for generating small keys.

6. Systems based on user specific keys / passwords

- **Kerberos**

In single sign-on systems such as Kerberos, passwords are limited to the use of characters that can be represented by 7 bit ASCII format. This password of any arbitrary length is converted into an encryption key that is stored in the Kerberos database which includes many steps including CBC mode of DES, these steps result in a CBC checksum, which is the key associated with this user's password. So we can view it as a hash function that maps an arbitrary password into a 64-bit hash code.
Both version of Kerberos are vulnerable to passwords attacks, the Authentication Server to the client includes information encrypted with a key based on clients passwords. If this data is comprised and suspected to proper attack, then it can reveal the password and hence the security of the system is compromised by gaining credentials from Kerberos. A counter measure for this situation is introduced in Kerberos version 5, a concept of pre authenticating, which is introduced to make password attacks more difficult but still it doesn't prevents the attacker much.
Also, it has long been known that Kerberos 4 Ticket Granting, Tickets, are susceptible to dictionary attacks, as they contain a constant string that can be used for compares, the string happens to be "*krbtgt*". It is possible to query a Kerberos server, provide a valid principle (user and Kerberos realm), receive a Ticket Granting Ticket, and decrypt the DES ticket using dictionary words for the key, if the phrase "*krbtgt*" exists in the decrypted packet then correct key is exposed.

- **PGP**

PGP (Pretty good privacy), system for standard email, requires pass phrases. Like many other systems, PGP uses passwords to control access; it uses the password to generate a 128-bit code using a hash function. As password can be of any length that's why it is called "pass phrase". Pass phrase is used to encrypt the user's copy of the secret key that is stored on user's computer hard drive and also user's file(s). It is recommended that these phrases should be complex sentences making no sense at all, for e.g. "666babyicecream". This sort of paradigm do help against the dictionary attacks, reverse dictionary attacks but the point is that the user can't remember long, complex, peculiar phrases (passwords), which provides room for various attacks, which can lead the attacker to read all the encrypted emails.

7. Conclusion

In this project we have studied common key and password generation. Many systems use passwords or similar information for generation of the cryptographic key. That is why attacks against keys sometimes become attacks against passwords. We studied different ways of generating passwords, weaknesses in user-chosen passwords and methods to protect against them. User-chosen passwords are most user-friendly and easy to remember, but they are not most secure. Due to many psychology reasons, users tend to choose passwords which could be recovered by various dictionary attacks. Various ways exists to prevent users from choosing too simple passwords. One of the best is proactive checking of selected passwords before introducing it to the database. This method allows to check the password against different dictionaries that attacker might use and evaluate how strong it is, if password is found too weak, user is suggested to choose another one. Different implementations of proactive password checker are used in most of the systems that allow user-selected passwords. In general, carefully designed proactive password checker will allow the system to be relatively safe against dictionary attacks.
Randomly generated passwords are generally much stronger, but it is more difficult to remember them. Plus, in some cases, information about the source of randomness might help attacker to reduce the search space. We studied different sources of randomness, which are used in generating random passwords and random keys. Random bits can be extracted from natural sources, like radio-active decay, noise, air turbulence in disk drives

and from system events, like interrupts from the I/O devices, user input latency, etc. Usually natural sources of randomness are more expansive because they require special hardware. But they are also more reliable and are used mostly in the systems that have higher security requirements.

If we take a look at the situation today, passwords remain the most popular form of authentication for low security systems. We use passwords to access our e-mail, our student account, different forums, e-commerce web-sites etc. Usually these systems leave password generation to the user, and it is up to the user, to generate truly random and "unbreakable" password or choose some string of characters, which can be recovered in a small amount of time.

Also, the structure of the authentication systems or technologies relies on one single entity, usually a master key, when deciphered compromises the security of the whole system. Depending on the nature of required security various architectures work with combination of two or more functionalities to make a system more secure, combining Pseudo Random Generator or/and True Random Number Generator.

References

1. Klein D., "Foiling the Cracker: A Survey of, and Improvements to, Password Security," Proceedings of the 2nd USENIX UNIX Security Workshop, pp. 514
2. Bishop M., Klein D., "Improving System Security via Proactive Password Checking," Computers and Security 14 (3), pp. 233-249 (Apr. 1995).
3. Spafford E., "Observing Reusable Password Choices," Proceedings of the 3rd UNIX Security Symposium, pp. 299-312
4. R. Ganesan and C. Davies., "A New Attack on Random Pronounceable Password Generators", Proc. 17th National Computer Security Conference", pp. 184-197, 1994
5. Bishop M., "Introduction to Computer Security"
6. M. Geisler, M. Krøigård and A. Danielsen. "About Random Bits", December 3, 2004.
7. L. Blum, M. Blum, and M. Shub. "A simple unpredictable pseudo-random number generator". In SIAM Journal on Computing, 15, May 1986.
8. C. Wright. "So You Need a Random Number Generator".
9. L'Ecuyer, P. and R. Proulx. 1989. about Polynomial - Time "Unpredictable" Generators. Proceedings of the 1989 Winter Simulation Conference. 467-476. IEEE Press: New York.
10. B. Schneier. "Applied Cryptology". Second edition, John Wiley & Sons, Inc.
11. RFC 1750, "Randomness Recommendations for Security".
12. Ritter, T. 1991. "The Efficient Generation of Cryptographic Confusion Sequences". Cryptologia. 15(2): 81-139.
13. Wikipedia. "Blum Blum Shub", In Wikipedia, the free encyclopedia. The Wikipedia Community, November 2004.
14. RSA laboratories, "RC4".
15. RSA laboratories, "SecurePC".
16. Wikipedia. "RC4Linear Congruential Generator", In Wikipedia, the free encyclopedia. The Wikipedia Community.
17. Scott R. Fluhrer, Itsik Mantin and Adi Shamir, Weaknesses in the Key Scheduling Algorithm of RC4. Selected Areas in Cryptography 2001.
18. N. Ferguson, B. Schneier. "Practical Cryptology". First edition, John Wiley & Sons, Inc.
19. Stephen K. Park and Keith W. Miller Random Number Generators: Good Ones Are Hard To Find Communications of the ACM, 31(10):1192-1201, 1988.
20. J. Pisharath, "Linear Congruential Number Generators", Newer Math, Fall 2003.
21. W. Trappe and L. Washington. "Introduction to Cryptography with Coding Theory". First edition.
22. Wikipedia. "Linear Congruential Generator", In Wikipedia, the free encyclopedia. The Wikipedia Community.
23. D. Davis, R. Ihaka, and P. Fenstermache, "Cryptographic Randomness from Air Turbulence in Disk Drives".
24. Wikipedia. "Geiger Counter ", In Wikipedia, the free encyclopedia. The Wikipedia Community.
25. Wikipedia. "Radioactive decay ", In Wikipedia, the free encyclopedia. The Wikipedia Community.
26. M. Rohe, "RANDy - A True-Random Generator Based On Radioactive Decay", 2003.
27. P. Mutaf, "True random numbers from Wi-Fi background noise", February'06 Paris, FR.
28. Y. Wang, H. Zhang, Z. Shen, K. LI, "Thermal Noise Random Number Generator Based on SHA-2 (512)", Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005.
29. D. C. Ranasinghe, D. Lim, S. Devadas, D. Abbott, P. H. Cole, "Random numbers from metastability and thermal noise".
30. J. Amenedo, R. McCue and, B. H. Simov. "Extracting randomness from external

resources", System Networking and Security Lab Hewlett-Packard Co.
31. Z. Gutterman, B. Pinkas and T. Reinman, "Analysis of the Linux Random Number Generator", March 6, 2006.
32. "Applied Cryptology". Second edition, John Wiley & Sons, Inc.
33. W. Stallings, "Network security essentials, application and standards".
34. LE Webmaster, "Linux exposed, Kerberos Security".
35. B. Schneier. ""Secret and Lies: Security in a network world"".
36. S. Garfinkel. "PGP: Pretty Good Privacy".