DRDO–IISc Programme on
Advanced Research in Mathematical
Engineering

Randomised attacks on passwords

(TR-PME-2010-11)

by
Manjesh Kumar Hanawal and Rajesh Sundaresan
Department of Electrical Communication Engineering, Indian Institute of Science,
Bangalore

12 February 2010

Indian Institute of Science
Bangalore 560 012

Randomised attacks on passwords

Manjesh Kumar Hanawal and Rajesh Sundaresan

12 February 2010

ABSTRACT

A brief survey of authentication methods in password-protected systems is first provided. Subsequently, features of a freely available software for cracking passwords is then discussed. A randomised procedure for attacking passwords, based on a tilted version of the password distribution, is then proposed. This tilt is a mechanism suggested by large deviations theory. After showing the asymptotic optimality of the proposed randomised scheme, an algorithm for generating these randomised attacks, based on the Metropolis algorithm, is then finally described.

Keywords: password, attacks, guessing, Markov chain Monte Carlo (MCMC)

1 Introduction

Passwords are still the preferred means to provide authentication in computer systems because of their simplicity. In all computer systems with password based authentication, user passwords are first enciphered using one-way functions like LanMan, DES, MD5, Blowfish, SHA, and then stored in a specific location. We refer to this enciphered password as hash. When a user claims access to the system through his password, it is enciphered using the same one-way function that generated the stored hash, and then matched with the stored hash. Authentication is granted if the hash matches the stored one. Usually, the one-way hash functions are publicly known. The encryption is strong if it is difficult to get access without entering the correct password. The password also should not be easily guessable. Below we will briefly discuss the password authentication schemes and attacks on them in Windows and Linux systems.

Windows based systems use the LanMan(LM), NTLM algorithms to generate the hash. The LM hashing algorithm first converts all ASCII characters to lower case and pads it with some fixed characters to make it 14 bytes in length. This padded password is split into two parts of 7 bytes each and enciphered separately using the DES algorithm. In principle, breaking this hash, which is the same as recovering the password, is the same as attacking passwords of character size 7. There are efficient probabilistic algorithms that exploit this weakness of LM hashing algorithm, an algorithm supported in all Windows operating systems for backward compatibility. Rainbow tables [1] introduced by Oechslin [1] based on Hellman's [2] time-memory tradeoff can efficiently attack the LM hashed passwords given the stored hash. Software based on rainbow table can be downloaded from http://ophcrack.sourceforge.net/.

Linux systems use the crypt() functions [3] to generate the hashes of the users' passwords. Before generating the hash, the Linux system adds a random string of fixed length (12-32 bits), called salt, to passwords. The salt can vary from user to user. Hash and its associated salts are stored along with other user related information in the path /etc/shadow. When a user enters a password, the corresponding salt is read from this location and appended to the input password before computation of hash. The benefits of precomputations are nulled by the salting technique because one precomputed table is required for every possible salt. For example, if the random string is of length 12 bits it requires 4096 precomputed tables to crack the password using above mentioned technique that was effective on the Windows system. Many Linux system use random string of length 32 bits or more, thereby making precomputation attacks infeasible. Unlike the windows system, there are no known weakness in the enciphering algorithms used in Linux system (except that some Linux machines use the traditional DES algorithm that restricts password length to the first 8 characters, or the MD5 algorithm [4]). Most of the well-known algorithms that attack Linux system passwords exploit poor choice of passwords by employing intelligent guessing attacks. One example is John the Ripper [5]; see section 2. To know more about other password cracking applications, visit http://sectools.org/crackers.html.

In this report we propose a randomised password attack based on some information theoretic ideas on guessing [6], [7], [8]. The candidate passwords are generated based on a tilted version of the original distribution. We prove asymptotic optimality of this technique and explain how password candidates can be generated using a Markov chain Monte Carlo method (see Diaconis [9] for a recent survey or Hastings [10]).

This report is organised as following. In Section 2 we explain the functionalities of John the Ripper software and explain its different modes in attacking a password. In section 3 we propose a guessing strategy that is asymptotically optimal. In section 4 we explain how strings with a tilted distribution can be generated using the Metropolis algorithm.

2 John the Ripper

John the Ripper (JtR) aims to help system administrators detect weak passwords. It works on both Windows and Linux systems. It supports many crypt(3) hash types commonly used in various versions of Linux systems. It has custom built functions to implement encryption algorithms that are optimized to particular system architectures. The latest version of JtR can be downloaded from Openwall project page http://www.openwall.com/. Let the directory in which JtR is installed be called John. To install JtR refer to instructions in file John/docs/INSTALL. JtR has the following modes: single crack, wordlist, external, and incremental.

In single crack mode JtR uses information on user name, home directory name, phone number, etc. that are stored in the file containing the hash in order to generate candidate passwords. Other candidate passwords are also generated using word mangling rules. New mangling rules can be specified, as given in John/docs/RULES. Wordlist mode uses the file John/run/password.lst that contains frequently used dictionary word passwords. Again, mangling rules will generate new candidate passwords from every word in this file. In external cracking mode, one can specify customised rules to generate candidate passwords. It is the incremental mode that we are interested in, and will be described shortly. See file John/docs/MODES to know more about these modes.

When JtR is executed on a hash file in its default mode, it first tries the single crack mode and then the wordlist mode. When both fail, it switches to the incremental mode. In this mode, JtR assumes no information about the hash, i.e., it assumes perfect secrecy [11, Sec. 2], and candidate passwords are generated based on the trigram statistics of ASCII characters. JtR has a precomputed table (John/run/all.chr) of ASCII characters arranged in a particular fashion for different passwords lengths. To understand the structure of this table let us fix length of passwords to say L and number of different possible characters to CSIZE. The following different matrices (refer to the function "inc_new_length" in file John/src/inc.c) are precomputed tables in JtR.

- A row vector of size CSIZE, containing all CSIZE characters of interest, and arranged in decreasing order of their frequencies of occurrence. Denote this row vector by $\mathbb{A}$.

- A 2-dimensional character matrix of size CSIZE $\times$ CSIZE. Each row is indexed by a character. In each row characters are arranged in the decreasing order of their frequencies of occurrence, when the previous character is the index for that row. Denote this matrix by $\mathbb{B}$.

- $L-2$ number of 3-dimensional matrices for positions $3, 4, \cdots, L$; each matrix is of size CSIZE $\times$ CSIZE $\times$ CSIZE. Here each row is indexed by character pair (a, b), and characters in that row are arranged in the decreasing order of their frequencies of occurrence, when the previous character pair is (a, b). Denote these matrixes by $\mathbb{C}[3], \mathbb{C}[4], \cdots, \mathbb{C}[L]$, respectively, for positions $3, 4, \cdots, L$.

Based on this precomputed table, the candidate passwords are generated as follows. Refer to function "inc_key_loop" in file John/src/inc.c file. To generate the password of length L, the first character is chosen from row vector $\mathbb{A}$. Let this character be denoted x_1. The second character is chosen from row with index x_1 in the 2-dimensional matrix $\mathbb{B}$. Let this character be denoted x_2. The third character is chosen from the row with index (x_1, x_2) in the 3-dimensional matrix $\mathbb{C}[3]$. The lth character is chosen similarly from the matrix $\mathbb{C}[l]$ with history appropriately updated. JtR generates these strings in the decreasing order of their preference and tests each full string as a candidate password.

The increment mode can run indefinitely depending on the length of the actual password and its character composition. This concludes our description of JtR.

In the following section we describe a method to generate strings using a "tilted distribution" and prove its asymptotic optimality.

3 Guessing with a tilt

First some notation. Let $(X_1, X_2, \cdots)$ be a sequence of random variables taking values in a finite set $\mathbb{X}$, and denote $X^n = (X_1, X_2, \cdots, X_n)$ for all $n \geq 1$. The bijective function

$$G_n : \mathbb{X}^n \to \{1, 2, \cdots, |\mathbb{X}|^n\}$$

is such that $G_n(x^n) = i$ means that the ith guess is the string x^n. The expected number of guesses is minimized when the guessing proceeds in the decreasing order of probabilities of strings [6]. In other words, for any $x^n, y^n \in \mathbb{X}^n$, if x^n has a higher probability of occurrence, $P_n(x^n) > P_n(y^n)$, then x^n is guessed before y^n, i.e., $G_n(x^n) < G_n(y^n)$. Here P_n denotes the probability mass function of strings on $\mathbb{X}^n$. We denote this optimal guessing function by G_n^*. Arikan [7] showed that expectation of G_n^* grows exponentially in n, and for an independently and identically distributed sequence of random variables he obtained the exponential growth rate to be

$$2 \log \sum_{x \in \mathbb{X}} \sqrt{P_1(x)},$$

i.e., the Rényi entropy of order 2. The Rényi entropy of order $\alpha > 0$ of distribution P_n is given by

$$H_\alpha(P_n) := \frac{1}{1-\alpha} \log \sum_{x^n \in \mathbb{X}^n} P_n^\alpha(x^n).$$

The above result of Arikan [7] was extended to Markov sources by Malone and Sullivan [12], to unifilar sources by Sundaresan [11], to a class of stationary sources by Pfister and Sullivan [13]. Our works [14]-[15] provide a unifying large deviations framework to all these results. The optimal limiting guessing exponent for a general source is given by the limiting Rényi entropy rate, if it exists, as follows:

$$\lim_{n\to\infty} n^{-1} \log \mathbb{E}_P[G_n^*(X^n)] = \lim_{n\to\infty} n^{-1} H_{1/2}(P_n).$$

For a proof of this statement, see [15].

Let us return to password cracking. We think of $\mathbb{X}$ as the alphabet of letters constituting the passwords. P_n is then the distribution on passwords of length n. JtR has simply precomputed the ordering under the assumption of a nonhomogeneous Markov source of memory order 2 and stored the ordering in the matrices described in the previous section. This will achieve the optimal guessing exponent. Instead, we now provide a simple randomised attack on the guessing exponent.

We first begin with the most natural attack and highlight its futility. Suppose that the correct password is x^n. If one generated passwords randomly and independently of one another, and according to the true distribution, then the number of guesses to hit x^n is a positive-integer valued random variable with a geometric distribution

$$\Pr\{G_n^P(x^n) = k\} = (1 - P_n(x^n))^{k-1} P_n(x^n), \quad k \geq 1,$$

so that its mean is

$$\mathbb{E}_P\left[G_n^P(X^n) | X^n = x^n\right] = 1/P_n(x^n).$$

Taking expectation this time with respect to X^n, we get that the expected number of guesses is

$$\mathbb{E}_P[G_n^P(X^n)] = \sum_{x^n} P_n(x^n) \times (1/P_n(x^n)) = |\mathbb{X}|^n,$$

which is not optimal.

Let us now generate guesses $y^n(1), y^n(2), \cdots$ according to the tilted distribution Q_n defined as

$$Q_n(x^n) = \frac{\sqrt{P_n(x^n)}}{\sum_{a^n \in \mathbb{X}^n} \sqrt{P_n(a^n)}}.$$

The random number of guesses needed, when x^n is the realisation, is $G_n^Q(x^n)$ which is a positive integer valued random variable having the geometric distribution with mean $1/Q_n(x^n)$. Performing the same computation as above, we get

$$\begin{aligned}\mathbb{E}_P[G_n^Q(X^n)] &= \sum_{x^n} P_n(x^n) \times (1/Q_n(x^n)) \\ &= \sum_{x^n} P_n(x^n) \left(\frac{\sum_{a^n} \sqrt{P_n(a^n)}}{\sqrt{P_n(x^n)}} \right) \\ &= \left(\sum_{x^n} \sqrt{P_n(x^n)} \right)^2 \\ &= \exp\{H_{1/2}(P_n)\}.\end{aligned}$$

Using this we get that the limiting exponent for the randomised and tilted guessing strategy is

$$\lim_{n\to\infty} n^{-1} \log \mathbb{E}_P[G_n^Q(X^n)] = \lim_{n\to\infty} n^{-1} H_{1/2}(P_n), \tag{1}$$

the optimal guessing exponent, and asymptotic optimality of this guessing strategy is proved.

There still remains the problem of generating sequences with distribution Q_n. To do this one needs to compute the partition function which is the normalising constant that makes Q_n a probability distribution. In the next section, we show how to avoid this computation, via Markov chain Monte Carlo (MCMC) methods.

In employing the MCMC method, the sequence of randomised guesses $y^n(1), y^n(2), \cdots$ does not constitute an iid chain, but is a Markov chain. In particular, $G_n^Q(x^n)$ does not have the geometric distribution.

However, we exploit the following fact. Suppose that the process of guesses $y^n(1), y^n(2), \cdots$ is an ergodic Markov chain with stationary distribution Q^n. Then, it is a well-known fact due to Kac [16] that the expected time for the first occurrence of a particular sequence x^n is

$$\mathbb{E}[G_n^Q(x^n)] = 1/Q_n(x^n).$$

This is all that is needed to validate the computations leading to (1). The next section identifies a procedure by which the sequence of guesses form a Markov chain whose stationary distribution is Q_n.

4 The Metropolis algorithm

We propose the use of the well-known Metropolis algorithm; we refer the reader to the tutorial survey by Diaconis on the Markov chain Monte Carlo revolution [9]. Our state space will be $\mathbb{X}^n$. We would like to generate a ergodic Markov chain $y^n(1), y^n(2), \cdots$ on this state space according to some transition probability matrix $K(a^n, b^n)$ such that Q_n is its stationary distribution. Note that

$$Q_n(x^n) = \frac{\sqrt{P_n(x^n)}}{Z_n},$$

where Z_n is the normalising constant that we would like to avoid calculating. The function $\sqrt{P_n(x^n)}$ will be assumed specified and easily calculable. This is indeed the case if our original source is by itself a Markov chain with known parameters.

Let $J(a^n, b^n)$ be any symmetric stochastic matrix with strictly positive entries. Suppose $y^n(i) = a^n$. The algorithm that tells how to generate the next guess $y^n(i+1)$ proceeds as follows:

The Metropolis algorithm

- Set $a^n = y^n(i)$.
- Sample b^n according to $J(a^n, \cdot)$.
- Compute the acceptance ratio $A(a^n, b^n)$ given by

$$A(a^n, b^n) = \frac{Q_n(b^n)J(b^n, a^n)}{Q_n(a^n)J(a^n, b^n)} = \frac{Q_n(b^n)}{Q_n(a^n)} = \frac{\sqrt{P_n(b^n)}}{\sqrt{P_n(a^n)}}. \tag{2}$$

- If $b^n \neq a^n$ and $A(a^n, b^n) \geq 1$, then set $y^n(i+1) = b^n$.
- Otherwise, if $b^n \neq a^n$ and $A(a^n, b^n) < 1$, then set $y^n(i+1) = b^n$ with probability $A(a^n, b^n)$.
- Otherwise, set $y^n(i+1) = a^n$.
- Increment i, and repeat the procedure.

Note that these calculations do not need the normalising constant Z_n since it cancels out in (2). It is easily verified that the resulting transition probability matrix is $K(a^n, b^n)$ given by

$$K(a^n, b^n) = \begin{cases} J(a^n, b^n) & \text{if } b^n \neq a^n, \quad A(a^n, b^n) \geq 1, \\ J(a^n, b^n)A(a^n, b^n) & \text{if } b^n \neq a^n, \quad A(a^n, b^n) < 1, \\ J(a^n, b^n) + \displaystyle\sum_{c^n : A(a^n, c^n) < 1} J(a^n, c^n)(1 - A(a^n, c^n)), & \text{if } b^n = a^n. \end{cases}$$

The generated Markov chain thus has transition probability matrix K. Moreover, the detailed balance equations

$$Q_n(a^n)K(a^n, b^n) = Q_n(b^n)K(b^n, a^n)$$

are also satisfied so that the generated Markov chain is reversible with equilibrium distribution Q^n.

Since the chain is an ergodic Markov chain, the distribution of $Y^n(i)$ converges to Q_n as $i \to \infty$. After a sufficient number of guesses are generated, which is something that depends on the convergence rate to equilibrium, the distribution of the generated $Y^n(i)$ will be close to Q^n.

5 Concluding remarks

We end this report with some final remarks on the Metropolis algorithm.

- The fundamental assumption is that the application of the hash function on a candidate password is expensive. The generation of password candidates themselves, i.e., sampling according to $J(a^n, \cdot)$ and the computations leading to acceptance or rejection is not.
- When we reject a generated sample, $y^n(i+1)$ is merely set to $y^n(i)$, a password candidate that was already tried without success. This generated sample for $i+1$ need not be tested, resulting in saved effort.
- When the new sample b^n is such that $A(a^n, b^n) \geq 1$, observe that b^n is more likely than a^n according to the source distribution, i.e., $P_n(b^n) \geq P_n(a^n)$. This sample is then tested.
- Finally, when the new sample b^n is such that $A(a^n, b^n) < 1$, then b^n is less likely than a^n. It is then tested (accepted for test) only with probability $1 - \sqrt{P_n(b^n)/P_n(a^n)}$. It is this prudence with which samples are chosen, exploration with caution, that helps save guessing effort.

Acknowledgements This work was supported by the Defence Research & Development Organisation (DRDO), Ministry of Defence, Government of India, under the DRDO-IISc Programme on Advanced Research in Mathematical Engineering. Fruitful discussions with Shri Rajesh Pillai and Shri ... of SAG, DRDO, are gratefully acknowledged.

References

[1] P. Oechslin, “Making a faster cryptanalytic time-memory trade-off,” in *Advances in Cryptology - CRYPTO 2003*, ser. Lecture Notes in Computer Science, vol. 2729/2003. Springer Berlin / Heidelberg, Oct. 2003, pp. 617–630.

[2] M. Hellman, “A cryptanalytic time-memory trade off,” *IEEE Trans. Inf. Theory*, vol. 26, pp. 401–406, 1980.

[3] *Linux man page for crypt(3)*, http://linux.die.net/man/3/crypt.

[4] X. Wang and H. Yu, “How to break MD5 and other hash functions,” in *Advances in Cryptology - EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494/2005. Springer Berlin / Heidelberg, May 2005, pp. 19–35.

[5] *John the Ripper password cracker*, http://www.openwall.com/john/.

[6] J. L. Massey, “Guessing and entropy,” in *Proc. 1994 IEEE International Symposium on Information Theory*, Trondheim, Norway, Jun. 1994, p. 204.

[7] E. Arikan, “An inequality on guessing and its application to sequential decoding,” *IEEE Trans. Inf. Theory*, vol. 42, pp. 99–105, Jan. 1996.

[8] N. Merhav and E. Arikan, “The Shannon cipher system with a guessing wiretapper,” *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1860–1866, Sep. 1999.

[9] P. Diaconis, “The Markov chain Monte Carlo revolution,” *Bulletin of the American Mathematical Society*, vol. 46, no. 2, pp. 179–205, Apr 2009.

[10] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[11] R. Sundaresan, “Guessing based on length functions,” in *Proceedings of the Conference on Managing Complexity in a Distributed World, MCDES*, Bangalore, India, May 2008; *also available as* DRDO-IISc Programme in Mathematical Engineering Technical Report No. TR-PME-2007-02, Feb. 2007. http://pal.ece.iisc.ernet.in/PAM/tech_rep07/TR-PME-2007-02.pdf.

[12] D. Malone and W. G. Sullivan, “Guesswork and entropy,” *IEEE Trans. Inf. Theory*, vol. 50, no. 4, pp. 525–526, Mar. 2004.

[13] E. Pfister and W. G. Sullivan, “Rényi entropy, guesswork moments, and large deviations,” *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2794–2800, Nov. 2004.

[14] M. K. Hanawal and R. Sundaresan, “Guessing revisited: A large deviations approach,” in *Proc. National Conference on Communications*, Guwahati, India, Jan 2009.

[15] ——, “Guessing revisited: A large deviations approach,” *DRDO-IISc Programme in Mathematical Engineering Technical Report No. TR-PME-2008-08, Dec.*, 2008, *available at* http://pal.ece.iisc.ernet.in/PAM/tech_rep08/TR-PME-2008-08.pdf.

[16] M. Kac, “On the notion of recurrence in discrete stochastic processes,” *Bulletin of the American Mathematical Society*, pp. 1002–1010, Oct. 1947.