Introduction

Password-Based Encryption derives is a cryptographic method that derives encryption/decryption keys from user passwords using a derivation function, as well as an iteration count of salt-hashing on the password to increase the computational cost of cracking the password (and ipso-facto the plaintext). However, weak passwords remain vulnerable to brute force attacks.

The objective of this report is to estimate the time required for such attacks based on various factors, using experimental measurements obtained from the Java implementation of the Password Based Encryption with MD5 and DES

Brief Description of The Procedure: PBEWithMD5AndDES, and Average Times of Encryption and Decryption

PBEs.java PBEsDecryptionMode.java

The first part of this experiment measures the average time taken The programs above are Java implementations of PBEWithMD5AndDES which is Java's security standard for Password Based Encryption that uses both Message-Digest 5 and Data Encryption Standard Algorithms.

The experiment requires a fixed byte array representation of our salt:

```
byte[] salt = {(byte) 0xc7, (byte) 0x73, (byte) 0x21,
       (byte) 0x8c, (byte) 0x7e, (byte) 0xc8, (byte) 0xee, (byte) 0x99};
```

(String Representation: 0xC773218C7EC8EE99)

As well as a fixed integer count for the number of times the hash function is applied over the password and salt to give us the encryption key.

```
int count = 2048
```

I performed this encryption and decryption for a total of 5 runs over all four passwords provided, and i also added extra functionality of timing the entire program, where (in nanoseconds) the start time is recorded…

```
long startTimeOfProgram = System.nanoTime();
```

The time taken to generate the key is also recorded…

```
long timeTakenToGenerateKey = System.nanoTime();
System.out.println("Key Generated! Time taken to generate key = "
               + (timeTakenToGenerateKey-startTimeOfProgram)
               + " nanoseconds!");
```

And the time to perform the entire encryption/decryption on a plaintext, for each password is recorded using System.nanoTime() java method to provide high-resolution timing for accuracy

```
System.out.println("Entire encryption program took " + (endTimeOfProgram - startTimeOfProgram) + " nanoseconds!");
```

Here are the recorded measurements:

Password	Avg. Encryption Time (ns)	Avg. Decryption time (ns)	Number of Runs
P@$$W0rD	42010933.4	40653066.6	5
thisismypassword	40747916.6	40099899.8	5
VeryLongP@$$W0rD	40900925.2	42193816.8	5
%O^t#2Fv0JUjVdRV2RW%	41613233.2	40577241.4	5

The measured encryption and decryption times are approximately 40 million nanoseconds (0.04 seconds) for all passwords. This is expected however, since password length does not significantly affect the cost of key derivation in PBE. This is because the password is first transformed into a 256-bit key, and then its key is used for encryption and/or decryption., The throughput of this does not depend at all on the value of the key.

FORMULAE

Brute Force Model

$Keyspace = C^N$
$Where\ C = size\ of\ character\ set$ (set of possible characters that can be used in making the password)
$N = Password\ Length$

Expected Guesses and Attack Time

In the best case, Expected guesses can be computed using
$Expected\ guesses = \frac{Keyspace}{2}$
$Attack\ Time = Keyspace \times Time\ per\ Guess$ OR $\frac{Keyspace}{Number\ of\ Guesses\ in\ a\ second}$

Brute Force Attack Estimation

Problem Formulation

Suppose an Attacker attempts a brute force attack on the password using the following information:

- Predefined plaintext,
- Resulting ciphertext,
- The Salt,
- Iteration count,

One can estimate the time required for a successful attempt, by first establishing the Keyspace (the number of possible character combinations)

(Please note that for this experiment, the set of case sensitive alpha-numeric (CSAN) symbols WILL include special case characters to make it 94 elements. I have also excluded the space bar as a character. Every other set is to be treated as usual

Attack Time and Guess Rate

Attack Time is calculated as a product of the keyspace and the time it takes to make a single guess (0.04). Since the attacker must repeat the encryption/decryption process for each password guess, we use a rounding off, of the same time as previously recorded, to represent the time it takes to guess a single password. (approx. 40,000,000 nanoseconds or 0.04 seconds per guess)

Which means that The system can perform approx. 25 guesses per second

Password	Length /N	Character Set /C	Key Space /C^N	Attack Time (years)
P@$$W0rD	8	CSAN	94^8 = 6095689385410816 = $6.1x10^{15}$	7737189
thisismypassword	16	Lowecase Alphabetic	26^{16} = 43608742899428874059776 = $4.36x10^{22}$	5531296664057 4
VeryLongP@$$W0rD	16	CSAN	94^{16} = 371574290834100916859450897 85856 = $3.72x10^{31}$	$4.71*10^{22}$
%O^t#2Fv0JUjVdRV2RW%	20	CSAN	94^{20} = 290106241131461823373062754 6741369470976 = $2.9*10^{39}$:	$3.7*10^{30}$

Here we see that an increase in key space leads to an increase in the attack time of the password. It is worth noting however, that in practice, brute force attacks will often succeed before the total search space has been explored [1]. On average, half the key space needs to be searched before the correct key is found
For the purpose of this analysis, I will use the worst case timing. Assume that the attacker must exhaust all possible combinations before a password is found

The Effect of the Iteration Count on the Attack Time

Variant 1 - No Password

Given a fixed password, and a fixed salt, we can observe the effect increasing the iteration count has on the attack time of the password. The salt remains the same as aforementioned (i.e 0xC773218C7EC8EE99). We start the observation for an iteration count from range 100-5000

Iteration Count	Time for Key Generation (s)	Total Encryption time/ Time per Guess(s)	Number of Guesses Per second	Attack Time (s)
100	0.035366542	0.036325083	27	$1.05*10^{38}$
500	0.030155042	0.037995125	26	$1.10*10^{38}$
1000	0.028120292	0.037258584	26	$1.08*10^{38}$
2000	0.027682333	0.042995583	23	$1.25*10^{38}$
3000	0.027365750	0.043302125	23	$1.26*10^{38}$
4000	0.027269625	0.044911333	22	$1.30*10^{38}$
5000	0.026925584	0.040376459	24	$1.17*10^{38}$

Although the measured execution times do not show a clear increasing trend, this is likely due to measurement noise, external factors such as runtime optimisation, and the relatively small computation time of the program. We can see to some extent, the relationship between iteration count and attack time is linear. Theoretically, Since the hashing is repeated once per iteration, we should see that the cost of key derivation increases linearly with the iteration count [2]

Variant 2 - No Password, No Iteration Count

Assuming the attacker must brute force both password and iteration count, I can take a worst case approximation of the attack time, by considering the upper bounds of the iteration counts range
Using a fixed password key space of 94^{20}, a fixed time per guess of 0.04s and an iteration count that ranges between 100 and 5000, (4900 possible counts) we would compute the new search space by taking a product of the key space, the upper bound of the count range, and the time per each guess
i.e Worst case Attack Time = Key Space x Time per each guess x Upper bound of iteration count

Iteration Count	Time for Key Generation (s)	Total Encryption time/Time per guess(s)	Number of Guesses Per second	New Attack Time(s)	Old Attack time (s)
100	0.035366542	0.036325083	27	$5.16*10^{41}$	$1.05*10^{38}$
500	0.030155042	0.037995125	26	$5.40*10^{41}$	$1.10*10^{38}$
1000	0.028120292	0.037258584	26	$5.30*10^{41}$	$1.08*10^{38}$
2000	0.027682333	0.042995583	23	$6.11*10^{41}$	$1.25*10^{38}$
3000	0.027365750	0.043302125	23	$6.15*10^{41}$	$1.26*10^{38}$
4000	0.027269625	0.044911333	22	$6.38*10^{41}$	$1.30*10^{38}$
5000	0.026925584	0.040376459	24	$5.74*10^{41}$	$1.17*10^{38}$

Since the attacker knows neither the password nor the iteration count, they must compute-for each character combination, every possible iteration count. The results show that this increases the previous total search space by a factor equal to the number of possible iteration counts.

Comparison of my Estimation to Other Established Estimations

Password	Time taken to brute force the password according to Security.org (years)	Time taken to brute force the password according to my findings(years)
P@$$W0rD	$9.13*10^{-4}$ (8 hours)	7,737,189
thisismypassword	34,000	55,312,966,640,574
VeryLongP@$$W0rD	$1*10^{12}$	$4.71*10^{22}$
%O^t#2Fv0JUjVdRV2RW%	$4.2*10^{19}$	$3.7*10^{30}$

There are a few plausible reasons as to why these times differ by such a wide margin.

- Security.org takes an average case search time, whereas my findings are built upon the worst case scenario where every character combination must be exhausted in order to get the correct password. Real password cracking tools use common password databases, dictionary attacks, [3] and even probabilistic password models [4] These dictionary attacks analyse common words/patterns in passwords meaning that the effective search space is much smaller, as opposed to my worst-case C^N approximation. For example, the password P@$$W0rD follows a common substitution pattern, and in such attacks this detail would lead to it being cracked much earlier

- Additionally, the assumed guess rate according to Security.org could differ. The rest of my measurements are made based on the time it took to encrypt the passwords (≈0.04 seconds), which depended on the processing time of my local machine. However Security.org may assume that the cracking system can perform billions of guesses per second, which is the case for high end GPUs, which are also more common within the cybercommunity

Conclusion

This study estimates the time required to perform brute force attacks against password-based encryption using experimental timing measurements. We see that password length and character set size significantly affect the size of the search space, while the iteration count increases the computational cost of each password guess, Comparisons with establish are dissimilar due to differences in hardware capabilities and attack strategies. But overall, the analysis demonstrates that long, randomly generated passwords combined with large iteration counts provide strong resistance against brute force attacks.

References and Citations

	Source
1	*Section 2.4.1 of Cryptanalysis of the SoDark family of cipher algorithms by* D. Marcus
2	Key derivation functions and their GPU implementation by O. Mosnáček
3	Understanding Dictionary Attacks in Cybersecurity
4	*A Study of Probabilistic Password Models* by J. Ma, W Yang, M.Liao and N. Li