A Note on the Security of Code Memo

Ruben Wolf
Fraunhofer-Institute for Secure Information Technology (SIT), Rheinstrasse 75, 64295 Darmstadt, Germany
ruben.wolf@sit.fraunhofer.de

Markus Schneider
Fraunhofer-Institute for Secure Information Technology (SIT), Rheinstrasse 75, 64295 Darmstadt, Germany
markus.schneider@sit.fraunhofer.de

ABSTRACT

Today, secret codes such as passwords and PINs are the most prevalent means for user authentication. Because of the constantly growing number of required secret codes, computer users are increasingly overtaxed. This leads to many problems in daily use, e.g., costs due to forgotten passwords in enterprises and security problems through bad password practice. Storing secret codes on mobile phones seems to be some kind of panacea to have secret codes always available since mobile phones are today's permanent companions. Code Memo is a software that is used on mobile phones to store secret codes in a safe way; it is provided as firmware on Sony Ericsson mobile phones. We assume that the intention of the Code Memo designers was to provide an ideal cipher system according to *Shannon's* classification, i.e., it leaves an adversary with uncertainty w.r.t. the correct decryption key. In this paper we show how to break Code Memo. For our attack, we have identified feedback channels in Code Memo that can be exploited for distinguishing correct master passwords from incorrect ones, and thereby, sieving candidates of master passwords. This weakness allows attackers in a realistic setting to identify the correct master password, and thus, to obtain all the stored passwords and PINs.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Code Breaking; D.4 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

Mobile applications, password management, security analysis

1. INTRODUCTION

Today, many computer applications, systems or services, e.g., Internet-based services or automated teller machines, require user authentication. In this context, the application of personal secrets, such as in the *password mechanism*, is the most widespread way to authenticate users. A typical computer user utilizes several applications, services, and systems that deploy user authentication based on the password mechanism. As a consequence, computer users have to manage too many secret codes in practice [5, 7, 8]. Typical Internet users may have an average number of 10–20 passwords. In this context, *managing* means, that they have to choose secret codes according to application-specific rules, memorize their codes, remember them correctly when they are needed, potentially change them within certain time intervals, and memorize their new codes again.

If users try to follow the hints for choosing good passwords —adequately long strings not guessable by others and consisting of a mix of lower and upper case letters, numbers and special characters— as they are given by security experts, they will hardly be successful in memorizing them. The result of following these hints is that users often forget their secret codes, and thus cannot start desired applications or access services. This has some negative impact especially for enterprises whose employees cannot be productive while they cannot start computer applications. Furthermore, enterprises have to deploy considerable helpdesk resources due to forgotten passwords [3, 4]. In order to solve this conflict between security and usability, users prevent forgetting their passwords in their daily activities by choosing passwords that can be remembered successfully, but can also be found out much more easily by adversaries [2]. Thus, while doing so, the security level typically decreases, which, in turn, may cause higher risks for users and enterprises[1]. At this end, users get in a situation that we call the *password dilemma*.

One possibility to get out of the password dilemma is to use technical helpers, such as password tools on personal computers or mobile phones. The application of password management on mobile phones has the advantage that secret codes are available in many situations, e.g., when being in the office or at home, at the bank's automated teller machine and at the supermarket checkout. This is due to the fact that mobile phones are today's permanent companions of human beings.

If secret codes are stored on mobile phones, then, of course, they should be stored in a secure way in order to prevent unauthorized persons from accessing them. Obviously, secret codes are highly-sensitive information and a multitude

MC'07 (Mobility'07), September 10-12, 2007, Singapore.

of them stored on a mobile phone is an attractive target for attackers.

Code Memo is a software for storing secret codes on a mobile phone. It is part of the Sony Ericsson mobile phone firmware. Unfortunately, we do not have any information about the cryptographic mechanisms that are applied by Code Memo. However, we assume that Code Memo encrypts and decrypts secret codes in a symmetric manner where the master password serves as encryption and decryption key. Obviously, Code Memo is a management tool for secret codes that tries —presumably— to provide properties of an ideal cipher system according to the classification given by *Shannon* in [6]. This means that given a known ciphertext, there remains an uncertainty with respect to the decryption key to the adversary if the plaintext is unknown to the adversary. In other words, if an adversary tries to decrypt the ciphertext by applying distinct keys, the decryption result seems to be a reasonable plaintext to the adversary for sufficiently many keys.

We have carried out a black-box analysis to Code Memo and detected feedback information channels that can be used to identify false decryption keys. Applying the same decryption key to other ciphertexts of the same user allows an attacker to sieve decryption keys. This means that it is possible to recognize incorrect master passwords. Since the key space of Code Memo is rather small, it is possible to uniquely identify the user's master password. As a consequence, it is possible to obtain all secret codes of a user that uses Code Memo.

In this paper, we show how an attacker can obtain a user's master password and all his secret codes. For this attack, it is sufficient to exploit one single weakness of several weaknesses we have found in Code Memo when analyzing its input/output behaviour. We explain the weakness that is exploited in our attack. Furthermore, we give a proof of concept on how to attack Code Memo by just applying freely available standard software.

2. OVERVIEW ON CODE MEMO

The internal mechanisms applied by Code Memo are unknown to the authors. However, it is possible to break Code Memo and obtain a user's master password and the stored secret codes. Since the attack requires only an observation of input and output parameters, we can restrict our overview to the basic properties and the input / output behaviour of Code Memo which is necessary to understand the attack.

A user when working with Code Memo has to select a 4-digit number of his choice which serves as his master password taken from the interval of integers $[0000, \ldots, 9999]$. This is shown in Figure 1. The master password in Code Memo is called *Passcode*.

After typing in the master password, Code Memo displays a checkword that serves as an indication to the user whether he has entered his master password correctly. An example is depicted in Figure 2 where the checkword has been set to *Lene*. This checkword has been initially set-up by the user. In case of a typo, the displayed checkword differs from the user's initially given checkword. If a typo has occured,

Figure 1: Entering the 4-digit master password

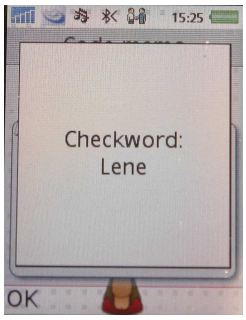

Figure 2: Displaying the checkword

the user receives another checkword string such as shown in Figure 3.

In the next step, Code Memo displays the user's list of applications and services for which he has stored his secret codes, as shown in Figure 4. Note that each secret code has to fulfill specific generation rules of the corresponding application. For instance, credit card PINs usually consist exclusively of 4-digit strings. Passwords can consist of characters taken from the set of lower and upper case characters, digits, and special characters.

When storing a PIN or a password, Code Memo recognizes the character sets from which the characters contained in the secret code are taken. Furthermore, Code Memo respects

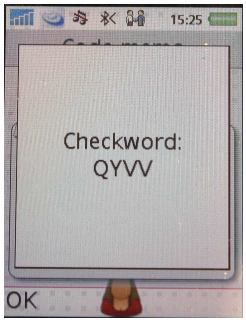

Figure 3: Displaying the checkword in case of a typo

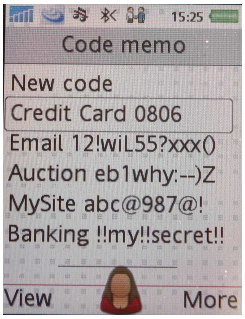

Figure 4: Displaying of the user's secret codes

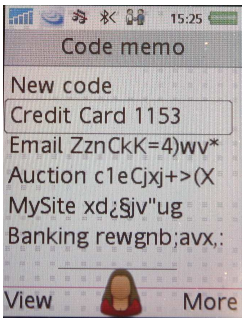

Figure 5: Return of wrong codes upon entering a wrong master password

the length of the given secret code. In case if an adversary is entering master password candidates, Code Memo will only display such results as seemingly correct secret codes that have the same length as the original secret code entered by the user and contain exclusively characters of the same sets as they are contained in the original secret code. The intention of this property is, presumably, to confuse an attacker by leaving him behind with uncertainty to decide whether he has found the correct master password or not.

This means, e.g., if a user stores his 4-digit credit card PIN in Code Memo, then upon giving in a wrong master password Code Memo returns a wrong PIN code also consisting of a 4-digit number. In the case of a stored password containing lower and upper case characters, digits, and special characters, the codes returned by Code Memo upon entering a wrong master password only contain elements from the same character sets. This property is depicted in Figure 5. Figure 5 shows the codes returned by Code Memo upon entering a wrong master password. Note that the returned codes correspond to the correct secret codes shown in Figure 4.

3. IDEA FOR ATTACK

3.1 Relevant Weakness

Let m be the user's master password. The weakness in Code Memo, that we have identified and that can be exploited to uniquely identify the user's master password m and to obtain all secret codes, is based on a property according to which Code Memo deals with special characters. The crucial point is that Code Memo deals with different input and output sets for special characters.

Let S be the input set for special characters. The output set for special characters is denoted by S'. This means when storing passwords in Code Memo, then users can exclusively enter characters that are contained in S. The special characters that are returned by Code Memo are taken from the set S'. As we have detected, it holds that $S \neq S'$ and $S \subset S'$. We introduce the difference set $D = S'/S$, or in other words, D is the complement to S with respect to $S' = S \cup D$ where $S \cap D = \emptyset$. The elements of S and D are shown in the Tables 1 and 2.[1] Note that the number of elements in these sets is $|S| = 15$ and $|D| = 17$.

Whenever, upon entering a candidate x for a master password, Code Memo returns a password candidate y that contains at least one character c for which $c \in D$ holds, it is immediately clear that $x \neq m$. This can be used by attackers to reduce the uncertainty whether the entered master password was correct or not. Considering one password candidate $y(x)$ upon entering x, there may remain some uncertainty if $y(x)$ contains no character c which is an element of D. However, this uncertainty can be reduced, or eliminated, if Code Memo still stores other passwords that contain special characters. The reduction, or elimination, of uncertainty is achieved through sieving as it is explained in Section 3.2.

In our opinion, this weakness is relevant for practice since users apply Code Memo to store their passwords and users are motivated by security experts —and also enforced by several applications— to choose passwords that contain special characters. Note that the weakness which is based on the way how Code Memo deals with special characters can even be exploited to obtain a user's PIN which exclusively consists of digits. This can be achieved by sieving as shown in the following.

3.2 Sieving Master Passwords

Let n be the number of passwords $pw_1, \ldots, pw_n$ containing special characters that are stored in Code Memo. For securing these passwords, the user has applied the master password m. An attacker with the intention to get a user's master password m and all his other secret codes enters master password candidates x_i for $i = 0, \ldots, 9999$ and observes the returned results $y_1(x_i), \ldots, y_n(x_i)$.

For each single password target pw_j with $j \in \{1, \ldots, n\}$, the attacker calculates sets X_j containing master password candidates according to

$$X_j = \{x_i \mid y_j(x_i) \text{ contains no character } c \in D\} \quad (1)$$

[1] At the first glance, it seems that the character ¶ is a member of S. Note that there is a special character ¶ that can also be entered. However, upon input of this character a line break is inserted, i.e., ¶ is not displayed as a symbol. If password candidates are returned as result to a wrong master password, Code Memo displays ¶ as a symbol. This can be easily distinguished from a line break. Thus, we have to assign the symbol ¶ to the set D.

Table 1: Input set for special characters S

set	S														
elements	.	,	-	?	!	'	@	:	;	/	(	)	*	+	#

Table 2: Difference set D

set	D																
elements	©	‗	%	<	®	&	§	$	¿	>	=	”	¥	£	¢	¶	¡

If all sets $X_1, \ldots, X_n$ are available, the attacker can reduce, or eliminate, uncertainty by sieving:

$$X = \bigcap_{j=1}^{n} X_j \tag{2}$$

Obviously, if the attacker has entered all values x_i, the set X is not empty. In the best case for the attacker, we have $|X| = 1$. Then, the element contained in X is identical to the user's master password and the attacker can immediately calculate all secret codes of the user. In typical realistic cases, X contains only one or very few elements. If $|X| > 1$, the attacker has some proposals for master passwords. However, in a small set of master password candidates the correct master password is quickly identified by calculating secret codes with these candidates and checking which password or PIN is correct. Typically, each service or application allows more than one attempt for giving in a secret code.

3.3 Probabilistic Assessment

In the following, it is our intention to analyze the extent of the reduction of uncertainty when an attacker enters master passwords in a brute force manner. This work is based on a simple probabilistic approach, that we have assumed for modeling Code Memo's properties when returning codes upon input of a false master password. Since we do not know the details for the calculation of the codes, our model is quite idealized. However, for typical cryptosystems the idealized model should fit well.

Based on our model, we show that the attacker's uncertainty with respect to master passwords can be dramatically reduced, even in cases that are very relevant for practice. There are even realistic cases in which the uncertainty can be completely eliminated.

Consider an attacker that enters master password candidates x. Since Code Memo works in a deterministic way, an attacker has to test each master password candidate x only once for trying to find a user's correct master password m. In a brute force attack, the user enters all potential master password candidates from 0000 to 9999. Among these, there are, obviously, the user's correct master password and 9999 false candidates. Every time when a user enters a false master password candidate and passwords containing special characters are returned, they can either contain elements from set D with a certain probability or not.

Let Z be a discrete random variable that represents the number of times the results returned by Code Memo contain at least one element from D upon entering all false master password candidates. Furthermore, let λ be the probability that the passwords calculated by Code Memo contain at least one element of D upon entering a false master password. Testing false master passwords and verifying whether they return elements of D can be modeled as *Bernoulli* trial with λ as success probability and $1 - \lambda$ as probability for failure, i.e., in this case returned passwords for a false master password candidate consist exclusively of elements taken from S. Modeling our experiment as *Bernoulli* trial is based on the assumption of statistical independence for repeated trials.

If the experiment is repeated M times for all false master passwords, i.e., $M = 9999$ times, then we obtain the expectation $\mathrm{E}[Z]$ and the variance $\mathrm{Var}[Z]$ as

$$\mathrm{E}[Z] = M \cdot \lambda, \tag{3}$$

$$\mathrm{Var}[Z] = M \cdot \lambda \cdot (1 - \lambda), \tag{4}$$

which are a well-known facts.

Now, let us have a look at λ. Obviously, λ depends on the number N of special characters that are displayed in total if an attacker enters a false master password. This depends on the number n of passwords a user has stored with Code Memo and the number of special characters contained in a decryption result for y_i. Let ν be the mean number of special characters contained in such a password candidate y_i. Then, we have $N = n \cdot \nu$. This yields the probability

$$\lambda = 1 - \left(\frac{|S|}{|S| + |D|}\right)^N = 1 - \left(\frac{15}{32}\right)^N = 1 - \left(\frac{15}{32}\right)^{n \cdot \nu} \tag{5}$$

Note that Equation (5) is based on the assumption that also the selection of N special characters can be modeled as a *Bernoulli* trial. Furthermore, we have to emphasize that Equation (5) exclusively holds for the 9999 false master password candidates. The correct master password never leads to results that contain elements of D.

Now, we have all required parts to have a closer look at the values of $\mathrm{E}[Z]$ and $\mathrm{Var}[Z]$ for some relevant cases. Therefore, we calculate $\mathrm{E}[Z]$ and $\mathrm{Var}[Z]$ for some pairs (n, ν). These are given in the Tables 3 and 4.

What do these values in the Tables 3 and 4 mean? In the

Table 3: Expectation E[Z]

E[Z]		n						
		1	2	3	4	5	6	7
ν	1	5311.96	7801.95	8969.13	9516.25	9772.71	9892.92	9949.27
	2	7801.95	9516.25	9892.92	9975.69	9993.87	9997.87	9998.75
	3	8969.13	9892.92	9988.07	9997.87	9998.88	9998.98	9998.99
	4	9516.25	9975.69	9997.87	9998.94	9998.99	9998.99	9998.99
	5	9772.71	9993.87	9998.88	9998.99	9998.99	9998.99	9998.99

Table 4: Variance Var[Z]

Var[Z]		n						
		1	2	3	4	5	6	7
ν	1	2489.98	1714.29	923.79	459.44	221.16	104.94	49.47
	2	1714.29	459.44	104.94	23.25	5.11	1.12	0.24
	3	923.79	104.94	10.91	1.12	0.11	0.01	0.001
	4	459.44	23.25	1.12	0.05	0.002	0.0001	6.1E-6
	5	221.16	5.11	0.11	0.002	5.9E-5	1.3E-6	3.0E-8

following, we give an interpretation of the results shown in these tables. Assume an attacker steals, or finds, a mobile phone where Code Memo has been used to store 4 passwords and the decryption results for these passwords contain 3 special characters on average. If the attacker carries out a brute force attack —potentially by using an apparatus as explained in Section 4 or any other more efficient approach— and enters all master password candidates from 0000 to 9999, then the attacker will get aware by observing Code Memo's results that averagely 9997.87 master password candidates from the 9999 false master password candidates can be recognized as false master passwords. So, there remains only an uncertainty of 1.13 master password candidates resulting from the set of false master passwords and the correct master password. This means, the after testing all master password candidates from 0000 to 9999, the attacker has an uncertainty of 2.13 master passwords, on average. This means that with probability of nearly 0.5 the attacker's first random choice from the sieved master passwords yields to the correct master password. Even if the first random choice is wrong, the attacker can take the next choice from its set of typically 2 or 3 sieved candidates. Note that the variance of Z is this small that the sieved set should not contain more candidates for master passwords in practice.

In a case in which Code Memo has been used to store 6 passwords and the decryption results contain averagely 3 special characters, an average of 9998.98 master password candidates from the 9999 false master password candidates can be immediately recognized. This means that it is very likely that sieving leads immediately to a single result, which is the correct master password. In this case, the attacker has immediately and completely eliminated all uncertainty.

The idealized model is useful as a theortical basis to see how much data, i.e., passwords, are necessary to reduce, or to eliminate uncertainty with respect to the correct master password. For setting up an attack scenario, however, there is no possibility to apply the expectation and variance as given in Equations (3) and (4) in a constructive way. The reason therefore is that there is no possibility in Code Memo —to our knowledge, at least— to control the average number of special characters that are returned upon entering false master passwords provided that this number is larger zero. Furthermore, the average number of special characters that are returned upon entering false master passwords for all inputs containing more than one special character is unknown to us. However, it is not the goal of this work to find out the average number ν of special characters in decryption results calculated by Code Memo. Nevertheless, having such values as given in Tables 3 and 4 is useful. They show how severe the detected weakness is for some reasonable values n and ν.

4. PROOF OF CONCEPT

In this section, we show how to attack Code Memo with standard software. Here, it was not our intention to optimize the attack. Instead, we want to show how easily the attack can be carried out by just applying standard tools, i.e., in principle, any normal user is able to carry out the attack. Of course, there is also the possibility to speed-up the attack by applying more specialized software tools. However, it is not our aim to explain how to make the attack more efficient. Nevertheless, experts having access to specialized software tools are able to speed-up the attack dramatically.

In order to show that our attack works successfully in reality, we have used an experimental setup as shown in Figure 6. Furthermore, we have stored some passwords in Code Memo under application of a master password selected by our own. The proof of concept illustrates how easily this master password can be found by attackers.

For our proof of concept, we require three devices as shown in Figure 6: a mobile phone (we have used a Sony Ericsson K800i), a personal computer, and a webcam. The webcam has been applied to capture the decryption results displayed by the mobile phone. The personal computer is used for driving the mobile phone, i.e., to enter master passwords candidates into Code Memo and to process the captured decryption results. Furthermore, the webcam is controlled

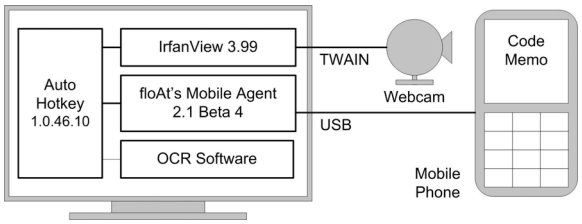

Figure 6: A simple architecture for the attack based on standard low cost components

by the personal computer via a TWAIN interface. For taking and storing pictures we have used the software *IrfanView Version 3.99*.[2] Driving the mobile phone is achieved by applying *floAt's Mobile Agent 2.1 Beta 4*.[3] The whole process, i.e., entering master passwords and capturing decryption results, is controlled by using *AutoHotKey Version 1.0.46.10*.[4] The pictures can then be postprocessed by using OCR software in order to identify special characters from set D.

In order to demonstrate the proof of concept, we have selected a random password setting; the passwords are selected according to security recommendations including lower and upper case letters, numbers, and special characters. We have to mention that there was no possibility for us to control the average number of special characters that are generated by Code Memo upon entering a wrong master password. The result calculated by Code Memo contains special characters with high probability if the password to be stored contains one or more special characters. In this case, the longer the password, the higher is this probability. Note that it was not the objective of this work to analyze the statistic output behaviour of Code Memo with respect to average numbers of special characters.

To show a case in which an attacker can eliminate all uncertainty with respect to the master password, we have decided to select a sample scenario with 7 passwords each consisting of 12 characters. Furthermore, the passwords are selected in such a way that they cannot be found in a typical dictionary attack and each password contained exactly 3 special characters. We emphasize that this should not be confused with the average number of special characters calculated by Code Memo upon entering master password candidates.

In our opinion, assuming a user that stores 7 passwords is not an exceptionel case. Today, a normal Internet user should have more than 7 passwords. The sample scenario is shown in Table 5. Furthermore, we have selected the PIN code 9876 as master password.

The attacker's first goal is to find the master password, and then, to get all other user secrets. When excuting the attack as described above with our sample scenario by entering all master password candidates $0000, \ldots, 9999$ in a brute force manner, it is possible to uniquely identify the correct master password, and thereby, to obtain all passwords of the sample scenario user. For this sample scenario, each entered master password candidate leads to at least one decryption result that contains an element from set D, i.e., an attacker can completely eliminate his uncertainty with respect to the correct master password. The only solution for the master password the attacker obtains trough sieving is 9876.

[2]See `http://www.irfanview.com/`.
[3]See `http://fma.sourceforge.net/`.
[4]See `http://www.autohotkey.com/`.

Table 5: Sample scenario

service	password
Email	w!A2z5:QX+7e
Auction	kF(6-3ZgK1.T
Banking	#?@9sKDaLjv2
Windows	7Go;O4's60M!
News	Ed1Nb8FwG)/-
Railway	A0z@+*3ijBX9
Bookstore	e@:/ETWkv2h1

If we omit the last entry (the Bookstore password) in our sample scenario and assume the other 6 passwords as given in the table above, then the attacker would obtain 4 master passwords candidates, i.e., 1127, 3629, 5378, and 9876. However, even if there is still some little uncertainty for the attacker, this can be easily eliminated in a second step by calculating all passwords for these master password candidates. Then, after at most three additional steps the correct master password and all other user passwords can be identified. Once the attacker has identified the correct master password, he can also obtain all other secrets that do not contain special characters, e.g., credit card PINs if stored together with passwords.

This proof of concept shows how easily an attacker can obtain a user's secrets if they are stored with Code Memo. Note that the software to execute this attack is no specialized hacking software. Furthermore, the software does not require in-depth security knowledge, which means that it can be carried out by any normal person.

5. CONCLUSION

We have identified a weakness in Code Memo that allows an attacker to obtain a user's master password and all his secret codes that are stored with Code Memo. The attack can be carried out under realistic conditions that are relevant in practice. In our opinion, this security weakness may have dramatic impact for the person who uses Code Memo, so that we cannot recommend using Code Memo anymore.

Since we detected the Code Memo weakness based on a black-box analysis, i.e., we do not know the internals of Code Memo, it is difficult —and potentially not effective— to give recommendations on how to repair Code Memo.

6. REFERENCES

[1] P. Ducklin. Simple advice for more sensible password use. http://www.sophos.com, Apr. 2006.
[2] W. Harrison. Passwords and Passion. *IEEE Software*, 23(4), July/August 2006.
[3] G. Hayday. It users in password hell. ZDNet UK News, Dec. 2002.
[4] G. Hayday. Counting the costs of forgotten passwords.

ZDNet UK News, Jan. 2003.
[5] SafeNet. 2004 Annual Password Survey Results. SafeNet (Inc.), http://www.safenet-inc.com, 2004.
[6] C. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4), 1949.
[7] Sophos. Employee password choices put business at risk. http://www.sophos.com, Apr. 2006.
[8] J. VanAuken. Review: Password Management: Grief Relief. Information Week, http://www.informationweek.com, Jan. 2006.