A Conceptual Framework for Assessing Password Quality

Wanli Ma, John Campbell, Dat Tran, and Dale Kleeman

School of Information Sciences and Engineering
University of Canberra, Australia

Summary
Password authentication is the most widely used authentication mechanism, and it will still be with us for many years yet to come. It is effective, simple, and accurate, with no extra cost. The strength of password authentication relies on the strength of the passwords. Good (or strong) passwords are essential for high level security. End user education and computerized proactive password checking play vital roles in ensuring good passwords. However, both demand clear, simple, and concise rules on what a good password is. It is not hard to find guidelines and advices on good passwords; but it is not so easy to find a clear, simple, and concise rule to be used for end user education and computer programs for proactive password checking. In this paper, we develop a theoretic framework on measuring password quality – password quality indicator (PQI). A PQI of a password is a pair $\lambda = (D, L)$, where *D* is the Levenshtein's edit distance of the password to the base dictionary words, and *L* is the effective password length. Based on PQI, we further simplify the rule for a good password to *at least 8 characters long, with at least 3 special characters plus other alphanumeric characters*.

Key words:
Password, Password Quality, Password Cracking, Computer Security, Levenshtein's Edit Distance,

1. Introduction

Authentication and authorization are the foundation of information security. Authentication is responsible for verifying that the person is really who he/she claims, and authorization is about assigning appropriate privileges to the person after the verification of his/her identity. Authentication is the first defense of the security operation of any information system.

There are 3 types of authentications [1, p 209]: (i) something the user knows, for example, password and PIN (personal identity number), (ii) something the user has, for example, physical keys, access cards, and smart cards etc., and (iii) something the use is – so called biometric authentication, such as voice recognition [2], fingerprints matching, and iris scanning etc.

Password authentication is simple, accurate, and effective. Although there are some concerns about weak passwords which lead to weak security; however, the weakness is not within the password authentication itself, but the choice of the passwords by human users. Physical keys or other physical devices, such as smartcards and access cards etc., are actually activated by passwords or PINs. They can be regarded as a hybrid authentication mechanism. Biometrics are useful to establish authenticity and for non-repudiation of a transaction. There has been a significant surge in the use of biometrics for user authentication in recent years because of the threat of terrorism and the Web-enabled world [3]. However, biometric authentication is a lot more expensive to implement, and yet not so accurate as password authentication. Furthermore, biometric features can be counterfeited, and they cannot 100% ensure authenticity or offer a guaranteed defense against repudiation. It is a good approach that different types of authentication systems working together to enhance security and performance [4, 5]. In summary, password authentication still is and will continue to be the working horse of information security. According to Saita [6], who reported the panel discussion at RSA 2005 conference, "*Password will be with us forever*", because "*We've got to make security simpler to use if it's going to be effective*", as suggested by the panel members.

The strength of password authentication relies on the strength of the passwords. We all understand the importance of choosing good or strong passwords [1]. Education and technical help are the keys for ensuring strong passwords [7]. Both education and technical help demand a clear and concise definition on what a good password is, in other words, a simple method of measuring password quality.

It is not hard to find guidelines or advices on choosing good passwords. For example, Yan et al [8] summarized that "*[A good] password should consist of mixed characters or special characters, and should not consist of words found in the dictionary*". Pfleeger and Pfleeger [1, pp 218-219] have these advices on choosing a good password "*Use characters other than just A-Z*", "*Choose long passwords*", "*Avoid actual names or works*", and "*Choose an unlikely password*". Similar advices can be found almost everywhere on security operation manuals,

[1] In this paper, we use good passwords and strong passwords interchangeably, and bad passwords and weak passwords interchangeably as well.

Manuscript received January 5, 2007.
Manuscript revised January 25, 2007.

policy papers, password security papers, and many web sites – e.g., [9]. After reading all these guidelines and advices, one may still not have a clear idea on what a good password is. Table 1 has 5 password examples. How do we judge these passwords? We know that Password 1 is not good because it is a dictionary word. How about the rest? Is Password 2 good or not? What about Password 3. And finally, how do we compare the quality of Password 4 and Password 5? Which one is better?

Table 1: password examples

	password
1	constitution
2	c0nstitution
3	c0ns+itution
4	akjuwfg
5	D$f9

Password quality is decided by the time required to crack the password. The longer time it requires, the stronger the password is. In this paper, we propose a means of measuring password quality – password quality indicator (PQI). The PQI of a password is a pair $\lambda = (D, L)$, where D is the Levenshtein's edit distance of the password to the base dictionary words, and L is the effective password length. The effective password length is the equivalent length of the password in the standard password format, which consists of only the 10 digit characters (0-9). From PQI, to avoid the costly operation of proactive password checking [10-12], we further develop a concise rule for choosing a good password: *a good password should be at least 8 characters long, with at least 3 special characters plus other alphanumeric characters*. The rule is easy to remember and easy to be checked by a computer program.

The paper is organized as follows. We first study password cracking strategies in Section 2, and then in Section 3, we develop the rationale of our password quality indicator theory. We conclude the paper with our future work in Section 4.

2. Cracking Passwords

There are many different types of password attacks, e.g., through remote logon, local logon, intercepting password hash on the network, and stealing password hash via SpyWare etc. [13, 14]. The goal of the attacks is the same: to gain unauthorized access by guessing the password. Regardless the types of password attacks, in essence, password cracking is about trying different character combinations until getting a match to the right password. In other word, password cracking is a trial and error process.

To effectively crack a password, some strategies have to be in place. The obvious combinations should be tried before the brute force enumeration of all possible password candidates.

Let's first list the cost of the brute force attack as the bench mark. Without the knowledge of the password makeup, the length of the password, and the character set used in the password, trying the combinations of all possible characters in all possible lengths is very costly. There are 93 printable characters[2], which can be used in passwords. These characters are:

- 26 lower case letters,
- 26 capital letters,
- 10 digit characters, and
- 31 special characters: `~!@#$ %^&*( )_-+= {}|[] ¥:"<> ?;',. /`

There are 93^n possible password candidates for the password length of *n*. Table 2 gives the number of possible password candidates for *n* from 3 to 10. From the table, we can see that combination explosion makes the trial and error process infeasible after the password length goes beyond 7 characters.

Instead of starting from this hard approach, dictionary words can be tried first. In this paper, we use dictionary words to stand for all common words, which include ordinary dictionary words, the names of people and places, commonly used phases, and movie names etc. There are many dictionary collections available, for example [15]. In this paper, for illustration purpose, we use the Fedora Core 5 English spell-checking dictionary as the example. The dictionary has 479,625 words. The shortest words are just 1 character long, and the longest word is "*pneumonoultramicroscopicsilicovolcanokoniosis*", 45 characters long. The average word length is 9.3 characters. Testing all these words by trial and error method only takes the effort at the same magnitude of enumerating all possible passwords of 3 characters long (Table 2), but we have more chances to succeed, as research suggested that dictionary words account for a large portion of passwords due to the memorability reason [13, 16].

If the dictionary words based cracking is not successful, the next best bet would be the variations to the dictionary words. Human beings are notoriously bad in handling meaningless random character strings. Passwords are for human beings to use. Therefore, a password has to be easily remembered. Small changes to the dictionary words fit perfectly into the situation.

[2] We ignore the possibility of using non-printable characters, such as white space, back space, and tab etc., in passwords. Using these characters or not does not change the validity of the argument in this paper.

Table 2: The number of possible password candidates of different password length

n	no of passwords	cracking time[3]
3	804,357	8.04 sec
4	74,805,201	12.47 min
5	6,956,883,693	19.32 hr
6	646,990,183,449	2.5 mon
7	60,170,087,060,757	19.34 yr
8	5,595,818,096,650,401	1,799.07 yr
9	520,411,082,988,487,296	167,313.23 yr
10	48,398,230,717,929,316,352	15,560,130.76 yr

For a word of *n* characters long, suppose that the derived words only have 1 character difference from the original word:

- There are *n+1* insertion positions: before the first character, in-between every 2 neighbor characters, and after the last character. For each position, there are *93* possible insertions (the *93* printable characters). All together, there are *93*(n+1)* possibilities.
- There are *n* deletions, which makes other *n* possibilities.
- There are *n* modifications. Each modification can take *92* possible choices – *93* printable characters minus the to-be-modified character itself. The operation gives the last *92*n* possibilities.

Putting the 3 operations together, for a word of *n* characters long, we have

$$93\times(n+1)+n+92\times n=93\times 2\times n+93 \qquad (1)$$

possible derived words. Each of the words is only 1 character different from the original word. If each of the derived words is 2 characters different from the original word, we can work out the total number of possibilities based on Formula (1):

- There are *93*(n+1)* words obtained by inserting 1 character into the original word. Each of the word is *n+1* character long. By Formula (1), *93*(n+1)* words give *[93*(n+1)]*[93*2*(n+1) + 93]* possible words, which are 2 character different from the original word.
- There are *n* words obtained by deleting 1 character from the original word. Each of the word is *n-1* characters long. Again, from Formula (1), they give *n*[93*2*(n-1) + 93]* possible words.
- There are *92*n* strings obtained by modifying 1 character from the original word. Each of the word is exactly *n* character long. They give *93*n*[93*2*n + 93]* possible new words, according to Formula (1).

All together, the number of password candidates which are 2 characters different from the original word is:

$$(2\times 93^2+93)\times 2\times n^2+(93^2\times 6-93)\times n+93^2\times 3 \qquad (2)$$

We still use the Fedora Core 5 Linux dictionary as the example. The dictionary has 479,625 words with average length of 9.3 characters. Using formula (1) and (2) on the average word length, we have Table 3.

Table 3: The number of possible password candidates by varying the dictionary words

no of diff	no password candidates	cracking time
0	479,625	4.79 sec
1	874,260,450	2.42 hr
2	1,686,357,413,595	6.51 mon

From Table 3, it is still hopeful to try 1 and 2 character's variations to the dictionary words, but there is no point to try 3 character's variations, as it is not as good as to start another round of enumeration from small character sets (below), for example, all lower case characters (26 characters), all upper case characters (26 characters), all lower case characters with digit characters (36 characters), or all upper case characters with digit characters (36 characters) Table 4.

Table 4: The number of password candidates of different password length (smaller character set)

n	26 characters		36 characters	
	no passwords	time	no. passwords	time
3	17,576	0.18 sec	46,656	0.47 sec
4	456,976	4.57 sec	1,679,616	16.8 sec
5	11,881,376	1.98 min	60,466,176	10.08 min
6	308,915,776	51.49 min	2,176,782,336	6.05 hr
7	8,031,810,176	22.31 hr	78,364,164,096	9.07 day
8	208,827,064,576	24.17 day	2,821,109,907,456	10.88 mon
9	5,429,503,678,976	1.74 yr	101,559,956,668,416	32.65 yr
10	141,167,095,653,376	45.39 yr	3,656,158,440,062,976	1175.46 yr

[3] In this paper, we assume that the encryption algorithm is crypt(3) on a standard Unix platform. The password cracking performance data is from a desktop computer (an Intel Pentium 4, 2.4 GHz, with Hyper-Threading turned off) running Fedora Core 5 Linux operating system. The platform can complete a crypt function call in approximately 10 microseconds. This performance implies a capacity to test up to 100,000 different passwords per second. The data will be different in different settings, but the assumption does not void the validity of the argument in this paper.

In summary, a likely path to crack a password will be, in the order of,

1. trying dictionary words,
2. trying 1 (and 2) character variations to the dictionary words,
3. trying to enumerate all possible password candidates which consist of a smaller character set, say just lower case characters or all lower case characters plus digit characters,
4. brute force enumeration of all possible password candidates with the full character set (93 characters).

3. Measuring Password Quality

The quality of a password depends on how long it takes, by trial and error method on possible password candidates, to find out the right match. The longer it takes, the better the quality is. Based on the analysis of the previous Section, we can measure the quality of a password by *how different it is from the dictionary words*, *how long it is*, and *how big the password character set is*. In this Section, we will first discuss these 3 measurements in details and then reduce them into 2 parameters of the password quality indicator (PQI). Finally, we develop a simple and concise rule on choosing good passwords.

How different is a password from the dictionary words? Levenshtein's edit distance [17] can accurately measure how different two strings are. This metric calculates the distance between two strings by counting the minimal number of single character manipulations required, such as an insertion, deletion, or modification, to make the 2 strings the same [18]. Table 5 gives some examples of Levenshtein's edit distances.

Table 5: Levenshtein's edit distances

string 1	string 2	distance
constitution	constitution	0
constitution	c0nstitution	1
constitution	constitutio	1
constitution	c0ns+itution	2
constitution	akjuwfg	11
constitution	D$f9	12

Levenshtein's edit distance can be used to measure how different a password is from all the base dictionary words: first, we line up all the dictionary words, and then, we check the Levenshtein's edit distance of the password against every single word on the line. The minimum distance is the distance of the password to the base dictionary words. Table 6 has some examples of the passwords we collected [13, 14] and their Levenshtein's edit distances against the Fedora Core 5 Linux dictionary.

Table 6: Levenshtein's edit distances of some sample passwords

password	closest dict word	distance
billabong	billabong	0
Basketball	basketball	1
phoenix09	phoenix	2
-boogie-	boogie	2
pass_word11	password	3
4dbabes1!	babes	4
Th1$1$$tup1d	chaetopod	8

How long is it? The length of a password is the number of characters in the password. The length plays a vital role in deciding how long it takes to crack the password. Table 2 and Table 4 list the number of all possible password candidates with different lengths and the time required to test all these candidates.

How big is the password character set? A password is made of characters. These character are called *composing characters*. There are from certain groups, e.g., all characters, low case alphabet characters, or digit characters. We call these groups *character sets*. From a computer's point of view, every character is the same; so is every character set. However, due to human preference, some are more preferred than the others. We artificially group the 93 printable characters into 4 sets:

- Character Set 1: 26 lower case letters – abcdefghijklmnopqrstuvwxyz
- Character Set 2: 26 upper case letters – ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Character Set 3: 10 digit characters – 01234567890
- Character Set 4: 31 special characters – `~!@#$ %^&*( )_-+= {}|[] ¥:″<> ?;’,./`

Character Set 1 is the most popular choice for passwords, and a significant portion of passwords only have characters from Character Set 1 [1, pp 214-216].

To measure the character sets used in a password, we propose *password complexity index (PCI)*. We assign PCI value 26 to Character Set 1, 26 to Character Set 2, 10 to Character Set 3, and 31 to Character Set 4. If a password contains a character from Character Set 1, the value 26 is added to the PCI of the password, so long and so forth. However, the value of each Character Set is only used once, i.e., the second and the subsequent character in the same Character Set do not add any extra value to the password PCI. Table 7 gives some examples of passwords and their PCIs.

Table 7: Passwords and their PCI values

password	password complexity index	
	component	value
billabong	26	26
Basketball	26+26	52
phoenix09	26+10	36
-boogie-	26+31	57
pass_word11	26+31+10	67
4dbabes1!	10+26+31	67
Th1$1$$tup1d	26+26+10+31	93

Effective Password Length: both password length and PCI have something to do with the number of possible password candidates, as shown in Table 2 and Table 4. Are they actually measure the same thing?

From Table 2 and Table 4, we can see that the number of 4 characters-long passwords with PCI 93 (Table 2) can be attained by roughly 6 characters-long passwords with PCI 26 (Table 4). Similarly, the number of 6 characters-long passwords with PCI 93 can be attained by 9 characters-long passwords with PCI 26. The same number of candidate passwords means the same level of difficulties, in terms of the time required to test all possible password candidates, in cracking the password. In a rough term, if we have lower PCI, we can choose longer passwords. To the theoretical extreme, we can just use a single character to make strong passwords, except for that these passwords will be extremely long.

To unify the measurements of password length and PCI, we propose the concept of effective password length. First, let's define standard password format.

Definition (Standard Password Format): if the characters of a password only draw from Character Set 3, i.e., only 10 digit characters (0-9), the password has the *Standard Password Format*. A password in the standard password format has PCI 10.

For example, passwords "125467" and "98456902" are in the standard password format, but "s125467", "8765t", and "ast+Ugh" are not. Standard formatted passwords are the base for measuring effective password length.

For a password **P**, which has PCI value *C* and length *m*, the number of all possible password candidates of the same format is C^m. To have the same number of password candidates in standard password format, which has PCI value of 10, we need to find out the length (*L*) of the password candidates in standard password format. Thus, we have:

$$C^m = 10^L \quad (3)$$

Therefore

$$L = m \times \log_{10} C \quad (4)$$

We call *L* the effective length of password P. The effective password length provides us a means to compare the lengths of passwords with different PCIs. Table 8 gives some example of effective password lengths. From the table, we can tell that, actually, passwords "akjuwfg" and "D$f9" are of the same strength.

Table 8: Effective password lengths

password	length	PCI / log PCI	effective length
billabong	9	26/1.14	10.26
Basketball	10	52/1.72	17.20
phoenix09	9	36/1.56	14.04
-boogie-	8	57/1.76	14.08
pass_word11	11	67/1.83	20.13
4dbabes1!	9	67/1.83	16.47
Th1$1$$tup1d	12	93/1.97	23.64
akjuwfg	7	26/1.14	7.98
D$f9	4	93/1.97	7.88

Password Quality Indicator: the *password quality indicator (PQI)* of a password is a pair $\lambda = (D, L)$, where *D* is the Levenshtein's edit distance of the password to the base dictionary words, and *L* is the effective password length. When *D>=3* and *L>=14*, we have a good password. *D>=3* means that the password is at least *3* characters different from the base dictionary words, and *L>=14* means that there are at least 10^{14} possible password candidates to be tried to crack the password.

The easiest way to achieve *D>=3* is to have *3* special characters (from Character Set 4) in the password. The requirement is easy to remember and also easy to implement with programs to check if a password meets the requirement. Of cause, by using *3* special characters to make *D>=3*, we do miss a number of password candidates which are *3* Levenshtein's edit distance units away from the base dictionary words. However, this simplified solution is justified for several reasons. First, *3* special characters guarantee the password has at least 3 units of Levenshtein's edit distance to the base dictionary words. Second, it is easy to remember for the end users. There is no need to explain to an end user on (i) what Levenshtein's edit distance is, (ii) how different a password should be from the base dictionary words, and (iii), more importantly, what all these base dictionary words are. Third, it is also easy to implement by computer programs to perform proactive password checking. It avoids the costly operation of traditional proactive password checking [10-12], which devotes most of its time in checking all possible dictionary words. As an example, there are 2×10^8 words listed at [15]. Finally, by forcing *3* special characters in the password guarantees an increase in the PCI value, and therefore, the effective length of the password.

Effective password length *L>=14* equals to *8* characters long for the real password in the suggest format.

For a password of this format, special characters give the PCI of the password value 31. The rest 5 characters of the password very likely have alphabet characters. To be on conservative side, let's assume the PCI of the password $31+26=57$, which is actually underestimated. The effective length *L=14* converts to the real password length *m*, by Formula (4):

$$m = \frac{L}{\log_{10} C} = \frac{14}{\log_{10} 57} = 7.95$$

In summary, as a simple rule for choosing good passwords, we propose that *a good password should be at least 8 characters long, with at least 3 special characters plus other alphanumeric characters*.

4. Conclusion and Future Work

Password authentication is effective, simple, and accurate, with no extra cost. It is the working horse of information security. Although there are growing interests on alternative authentication mechanisms, such as smartcards and biometrics etc., password authentication is still the pivotal of authentication regimes. This is due to the complexity and less accuracy nature of the alternative authentication methods. It is suggested that better security can be achieved by combining password authentication with the alternative authentication methods.

The strength of password authentication relies on the strength of the passwords used. A good password is absolutely essential in achieving high level of security. However, research shows that there still are many weak passwords in daily use on real life systems, although much effort has be invested in educating end users and computerized proactive password checking. We believe this is due to the lack of a simple and concise rule on what a good password is.

In this paper, we first studied password cracking strategies. Based on the study, we developed a theoretic framework for measuring password quality – password quality indicator (PQI). The quality of a password is in proportion to the time required to crack it by the trial and error method. Three aspects of the password decide the time required: (i) how different the password is from base dictionary words, (ii) how long the password is, and (iii) how big the password character set is. The first aspect is measured by Levenshtein's edit distance of the password to the base dictionary words. The last two aspects are all related to the number of possible password candidates to be tried by the trial and error method. By introducing the concepts of standard password format and effective password length, we can unify them into a single metric: effective password length. The PQI of a password is a pair $\lambda = (D, L)$, where *D* is the Levenshtein's edit distance of the password to the base dictionary words, and *L* is the effective password length. A good password should have *D>=3* and *L>=14*. This requirement can be simplified as *at least 8 characters long, with at least 3 special characters plus other alphanumeric characters*. Effective password length *14* (*L=14*) can be attained by *8* characters-long password in this good password format. Although there are good passwords which cannot be included in this format due to the restriction of 3 special characters, the simplification is justified for the purpose of end user education and computerized proactive password checking. In either of the situations, simplicity is the key to success.

We have 2 tasks to pursue as the next step of our research. First, we will test the effectiveness of the simplified rule in end user education and also proactive password checking, and second, we will study the impact of the simplified rule on password memorability.

Acknowledgment

This research work is supported by the divisional grants from the *Division of Business, Law and Information Sciences, University of Canberra, Australia*, and the university grants from *University of Canberra, Australia*.

References

[1] Pfleeger, C.P. and S.L. Pfleeger, *Security in Computing*. Third ed. 2003: Prentice Hall.

[2] Tran, D., M. Wagner, et al., *Fuzzy Methods for Voice-Based Person Authentication.* IEEJ (Institute of Electrical Engineers of Japan) Transactions on Electronics, Information and Systems, 2004. **124**(10): p. 1958-1963.

[3] Bolle, R.M., J. Connell, et al., *Biometrics 101, IBM Research Report*. 2002, IBM: IBM T. J. Hawthorne, New York, USA.

[4] Namboodiri, A.M. and A. K. Jain. *On-line Script Recognition*. in *the Sixteenth International Conference on Pattern Recognitio*. 2002.

[5] O'Gorman, L., *Comparing Passwords, Tokens, and Biometrics for User Authentication.* Proceedings of the IEEE, 2003. **91**(12): p. 2021-2040.

[6] Saita, A. *RSA 2005: Passwords at the breaking point*. 2005 [cited 14 January 2006]; Available from: http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1059143,00.html.

[7] Conlan, R.M. and P. Tarasewich. *Improving interface designs to help users choose better passwords.* in *CHI'06: CHI '06 extended abstracts on Human factors in computing systems*. 2006.

[8] Yan, J., A. Blackwell, et al., *Password Memorability and Security: Empirical Results.* IEEE Security & Privacy, 2004. **2**(5): p. 25-31.

[9] Wikipedia. *Password strength*. [cited 14 January 2007]; Available from: http://en.wikipedia.org/wiki/Password_strength.
[10] Cisneros, R., D. Bliss, and M. Garcia, *Password auditing applications.* Journal of Computing in Colleges, 2006. **21**(4): p. 196-202.
[11] Blundo, C., P. D'Arco, et al. *A novel approach to proactive password checking*. in *International Conference on Infrastructure Security (InfraSec 2002)*. 2002. Bristol, UK: Springer.
[12] Yan, J. *A Note on Proactive Password Checking*. in *ACM New Security Paradigms Workshop*. 2001. New Mexico, USA.
[13] Campbell, J., D. Kleeman, and W. Ma. *Password Composition Policy: Does Enforcement Lead to Better Password Choices?* in *17th Australasian Conference on Information Systems (ACIS 2006)*. 2006. Adelaide, Australia.
[14] Campbell, J., D. Kleeman, and W. Ma, *The Good and Not So Good of Enforcing Password Composition Rules.* Information Systems Security (accepted for publication).
[15] A.R.G.O.N. *Word Lists*. [cited 15 January 2007]; Available from: http://www.theargon.com/achilles/wordlists/.
[16] Bryant, K. and J. Campbell. *An Empirical Study of User Practice in Password Security and Management*. in *Australasian Conference on Information Systems*. 2005. Sydney, Australia.
[17] Levenshtein, V., *Binary codes capable of correcting deletions, insertions, and reversals.* Problems in Information Transmission, 1965. **1**: p. 8-17.
[18] Stephen, G., *String Searching Algorithms*. Lecture Notes Series on Computing. Vol. 3. 1994: World Scientific Publishing.

Dr. Wanli Ma is a Lecturer of School of Information Sciences and Engineering, University of Canberra. His research interests include computer security (intrusion detection, biometrics, and computer forensics) and multiagent system (system structure, applications, and agent based software engineering). Wanli Ma also have 6 year's first hand experience in running IT infrastructure and IT security operations.

John Campbell is Associate Professor of Information Systems, School of Information Sciences and Engineering, University of Canberra, Australia. His research expertise includes topics in information security, IT governance, IT investment evaluation, virtual communities, and organizational communication. A major theme throughout his work is how users interact through information systems in the social world and, in particular, the ways in which organizational decision-making and community interaction are enacted through collaborative technologies.

Dat Tran received his B.Sc. and M.Sc. degrees from University of Ho Chi Minh City, Vietnam, in 1984 and 1994, respectively, and his Ph.D. degree in Information Sciences and Engineering from University of Canberra (UC), Australia in 2001. He is currently senior lecturer in software engineering at UC. Dr. Tran has published about 90 research papers in biometric authentication, hidden Markov model, fuzzy modelling, spam email detection, language identification and cell phase recognition. Dr. Tran has been awarded about 10 research grants, and has served as reviewer for IEEE Transaction on Fuzzy Systems, IEEE Transaction on System, Man and Cybernetics, Pattern Recognition and Bioinformatics journals. He is IEEE senior member and WSEAS Australia Chapter board member. His biography has been included in Marquis Who's Who in the World and in Sciences and Engineering.

Dale Kleeman is a Senior Lecturer in Information Systems in the School of Information Sciences and Engineering, University of Canberra. He has extensive experience, both inside and outside of academia, in information systems audit and information systems management, and has been active in university governance. His current research interests include information security and applications of soft systems methodology.