Robust Two-factor Smart Card Authentication

Omer Mert Candan
Sabanci University
Istanbul, Turkey
mcandan@sabanciuniv.edu

Albert Levi
Sabanci University
Istanbul, Turkey
levi@sabanciuniv.edu

***Abstract*—Being very resilient devices, smart cards have been commonly used for two-factor authentication schemes. However, the possibility of side-channel attacks renders private data stored in the cards vulnerable to compromise. With this in mind, we propose an authentication protocol that incorporates a second factor, which is as a password, in addition to the smart card. The scheme is aimed to withstand most common security breaches as well as compromised smart card scenarios and offline dictionary attacks on the passwords. Details of a reference implementation are also given along with performance evaluation of the proposed protocol comparing to the literature. Performance analyses show that the proposed protocol outperforms existing solutions in the literature. Moreover, the computational cost of the proposed protocol is less than 2 seconds on our reference implementation that uses commercially available smart cards.**

Keywords—Two-factor Authentication; Remote Login; Smart Card; Offline Dictionary Attack; Elliptic Curve Cryptography

I. INTRODUCTION

Online services have been more and more prevalent in daily life. Internet users use remote servers all around the world to store their information, even private information. As a standard security measure, remote servers are expected to deny access to users that fail identification and authentication processes. The authentication usually depends on a pre-determined password that only the legitimate user is supposed to know. This introduces single point of vulnerability to whole mechanism. In the event that the password is compromised, the remote server will not be able to deny access to imposters. To remedy this, two-factor authentication protocols have been devised. In this paper, we propose such a protocol that utilizes passwords and smart cards together. Our protocol is a robust one such that it resists card stealing attacks or offline password guessing attacks.

In recent years, many different smart card based authentication schemes have been proposed [3, 5, 6, 7, 8]. Some of these protocols have been developed as an improvement to a previous protocol that has been found out to be insecure or inefficient [9, 10, 11]. Later, some of the improved protocols have been shown to have security issues themselves and enhanced versions of previously improved schemes have been brought to attention [12]. This long chain of design issues and solutions has shown that the security of an algorithm or a scheme depends heavily upon the security of the components that is built on.

Smart cards have been known to be safe to store private information, thus have been a trusted tool in two-factor authentication designs. However, side-channel attacks [1, 2] on smart cards have broken the assumption that the contents of a smart card are externally inaccessible. Since then, the protocols that relied on the card to store sensitive information have been found out to be weak against offline dictionary attacks on stolen smart cards. Offline dictionary attacks are possible to launch, since the password space is usually small for efficiency (easy to remember, easy to type). We accept the possibility that data stored in a smart card is extractable. On the other hand, we still assume that a random number generated on-the-fly cannot be revealed; otherwise the device would not be different than a white box. We base our protocol on this assumption, thereby, do not store any sensitive information on the card and use the card in the process of generating random numbers and utilizing them in the protocol run. Moreover, we aim that our proposed protocol would be (i) lightweight, (ii) resisting against common security attacks such as offline dictionary and replay attacks, (iii) feasible to be implemented on commercially available smart cards with standard features and capabilities. Our protocol is based on other Elliptic Curve based schemes [3, 5, 6]. The key difference between our protocol and previously presented ones is that our protocol is simpler, and operation-wise is comparable with other protocols. Additionally, we provide a reference implementation with actual timings of a run of the protocol.

The rest of this paper is organized as follows. In the next section (Section II), we give the details of our proposed robust authentication protocol. Section III covers security analysis of the proposed protocol. This analysis includes most of the common security attacks and how the proposed protocol is supposed to thwart them. Consecutively in Section IV, details of a reference implementation are presented. Runtime measurements of the implementation are given in Section V and finally Section VI concludes the paper.

II. OUR PROPOSED PROTOCOL

The proposed protocol aims to authenticate users to a remote server via smart cards. A user is assumed to have a unique ID, a password and a smart card. At the registration phase, which is performed only once, the user (the terms client and user are used interchangeably) and the server agree on multiple parameters while exchanging messages over a secure channel. When a client wishes to log in (authenticate) to the remote server, not only ID and the password should be known, but the smart card should also be present. Upon receiving ID, password pair; the smart card and the server communicates to verify each other. If the verification is successful and the user is authentic, then both sides can calculate a secret key that is supposed to be the same on both ends. The secret key can be utilized for further secure communication and the protocol ends after the derivation of the symmetric key.

A. Elliptic Curve (EC) and EC Diffie-Hellman (ECDH)

In this subsection, the well-known Elliptic Curve (EC) concept and Elliptic Curve Diffie-Hellman (ECDH) protocol are overviewed.

EC is a curve defined by an equation of the form:

$$y^2 = x^3 + ax + b$$

where a and b are real numbers. ECDH is a key agreement scheme that allows two parties to generate a secret value over public channel. ECDH depends on the idea that it is very hard or infeasible to solve EC Discrete Logarithm Problem (ECDLP): given two points P and Q on the curve such that $Q = k.P$.

In ECDH, both parties select their random secret values, (k_1 and k_2) and perform multiplication on the same point (P).

$$Q_1 = k_1.P$$
$$Q_2 = k_2.P$$

Then, they exchange Q_1 and Q_2. After the exchange, they calculate the shared secret as follows.

$$S_1 = k_1.Q_2$$
$$S_2 = k_2.Q_2$$

In our protocol, ECDH key agreement scheme will be used to create a temporary shared key in the login phase of the authentication protocol.

B. Registration Phase

The communication in this phase is assumed to be over a secure channel. The user, u_i determines a unique ID_i and a password PW_i. These are the inputs to the smart card. In the card, a random number, b, is generated and this number is kept secret throughout the lifetime of the card.

The server generates a secret random number, s, in the registration phase of the first user of the system. The same number is kept as server's secret and used for the remaining users that will register later. The server also chooses an elliptic curve, EC, a symmetric cipher algorithm $C(\cdot)$ and a secure hash function $H(\cdot)$ as system parameters.

The flow of registration phase is given in Fig. 1. In this phase, the user sends ID_i to the server. Upon receiving ID_i, the server checks if the ID is in use by another user. If the received ID is not unique, the registration phase is terminated.

In the reply message, the server specifies hash and encryption functions to be used for the entirety of the authentication protocol.

The user creates a large secret random number b and concatenates this number with the password and finally hashes it. The output of the hash operation is sent to the server. Meanwhile, user stores b, EC, $C(\cdot)$ together with her ID in the smart card, SC.

Obtaining the hash from the user, the server hashes its secret s concatenated with client's ID_i then applies XOR operation with the hash received from the user. Finally, the result A_i, is stored along with ID_i on the server side.

If the user wants to change the password, then a secure channel should be reestablished. This can be achieved by following the protocol and reaching to a shared session key between the user and the server. The user u_i creates a fresh random number b_{new} and sends this number along with $H(b_{old}||PW_{old})$ and PW_{new} to the server. The server recalculates A_i and then replaces the old value with the newly calculated value.

$$A_i' = A_i \oplus H(b_{old}||PW_{old}) \oplus H(b_{new}||PW_{new})$$

If the server needs to perform key rotation, i.e. change the secret number s_{old} with s_{new}, then all A_i must be recalculated as shown,

$$A_i' = A_i \oplus H(s_{old}||ID_i) \oplus H(s_{new}||ID_i)$$

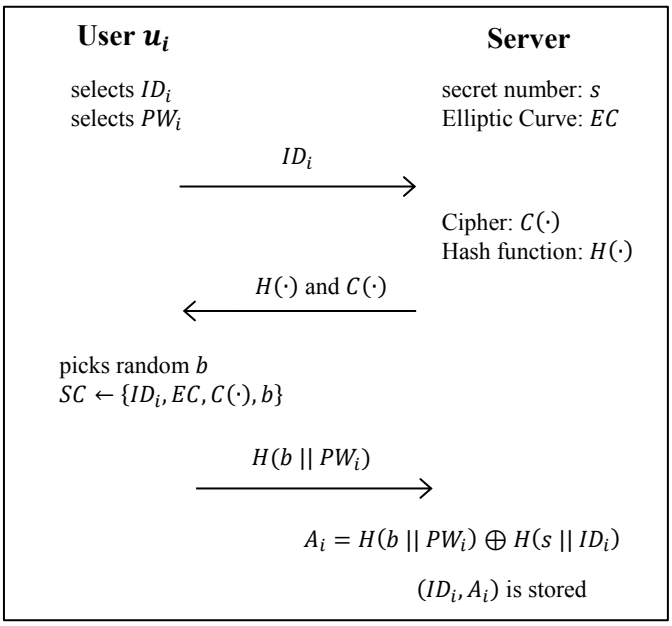

Fig. 1. Registration Phase

C. Login Preparation Phase

During login, the communication channel is considered to be insecure. The login is done in two phases: (i) login preparation and (ii) verification phases. The first one, login preparation phase is depicted in Fig. 2. The user u_i provides ID_i and PW_i to the smart card. A random point P in EC is chosen together with a random number d_u. Q_u is calculated by multiplying P with d_u, and sent to the server with P. The server chooses a random d_s and calculates Q_s by multiplying P with d_s and a shared key SK by multiplying Q_u with d_s. Client sends her ID and a timestamp encrypting it with freshly created SK. The server decrypts this message and this concludes the login preparation phase of the protocol.

D. Verification Phase

Verification phase, shown in Fig. 3, is the actual phase that the client is authenticated to the server. At this phase, the server and client switches to a modified version of the shared key that they generated in the previous phase. This new key is calculated as shown below.

$$SK' = SK \oplus H(b \,||\, PW_i)$$

With the help of this new key, the server sends an encrypted random challenge to the user and expects to see the same value in the next message from the user. In addition to responding to server's challenge, client puts its own random challenge in the same message. Naturally, the client will authenticate the server, if the server can successfully respond to the challenge. Responses of the challenges (messages 5 and 6) will have their own timestamps in order to prevent replay attacks.

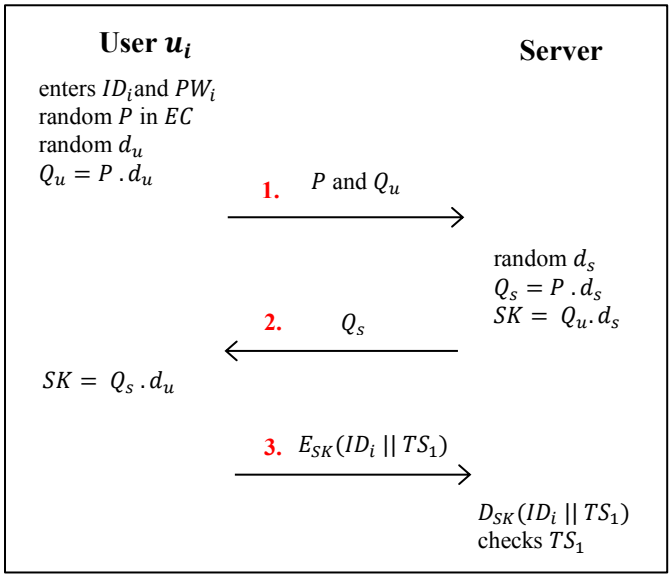

Fig. 2. Login Preparation Phase

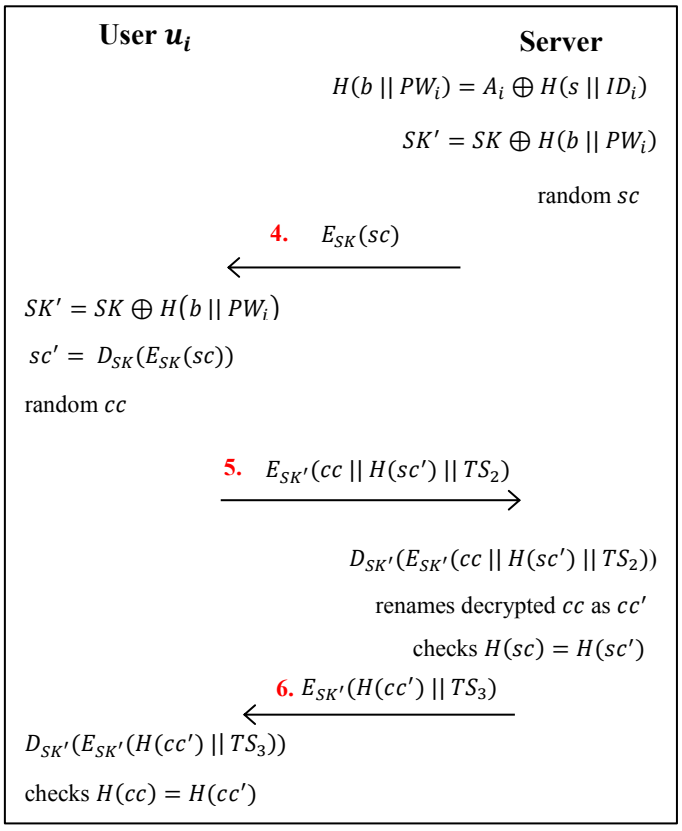

Fig. 3. Verification Phase

After the end of the verification phase, if both parties make sure about the identity of each other, they are individually able to calculate the same session key as show below. This session key can be used to secure further communication.

$$K_{session} = H(\, SK' \,||\, H\,(cc \,\oplus\, sc) \,||\, TS_2 \,||\, TS_3\,)$$

III. Security Analysis

We assume that the adversary, denoted with A in the subsequent discussions, has full control of the communication channel. Therefore, the attacker can eavesdrop, block, modify and inject packets into the insecure channel.

The second assumption is that the hash function we used is a cryptographically secure one-way hash function. It has pre-image resistance property so that it is not easy to find the input of the function by analyzing its output. Moreover, the symmetric cipher used is a computationally secure one.

Third assumption is that the attacker may compromise only one of the two components: *smart card* or *password*. This assumption is a reasonable one since if one obtains both, then he/she has all the credentials that a legitimate user has.

In the following subsections, we discuss possible security threats and how they are prevented.

A. Offline Dictionary Attack

Offline password guessing is a type of attack scenario in which the adversary A has the control over the smart card of u_i. A can extract secret information from the card, which is the user's secret random number b. Then, in order to complete the attack, A must learn PW by trying possible passwords from a dictionary in offline manner using the protocol messages. In order to obtain the protocol messages, A can either

1. observe captured messages from previous runs of protocol, or
2. initiate a new run with the server and try to find more from the incoming message.

In both cases, A first tries to obtain $H(b \,||\, PW)$, which is equal to $SK' \oplus SK$. Then, A launches a brute force attack on H(b || PW) by trying different PW′ values from the dictionary until he finds a correct PW′ such that, $H(b \,||\, PW') = H(b \,||\, PW)$.

In the first case mentioned above, A has a bunch of messages which were encrypted with either SK or SK'. In this case, A needs to find out both SK and SK' in order to obtain a test base (i.e. $H(b \,||\, PW)$) for password guessing. However, she does not know SK since she did not initiate the previous protocol runs. A can try to find out SK only by trying to break ECDH messages of the corresponding login preparation phase. However, this requires the attacker to solve ECDLP problem, which is computationally infeasible. For learning SK', A only has messages, which are encryptions of random numbers. These do not help her either since we assume the use of secure symmetric ciphers. Thus we conclude that without SK and SK', A cannot launch an offline dictionary attack.

For the second case mentioned above, A initiates a protocol run with the server and they calculate SK during the login preparation phase. Then, they continue with message 4 of the

verification phase. Message 4 contains $E_{SK'}(sc)$. A can try to calculate SK' by trying different PW' values by checking the equality $SK' = SK \oplus H(b \,||PW')$. Then for each SK', A can decrypt message 4 to find sc. However, since sc is a random number, A cannot be sure that the decrypted value is equal to sc. Thus A cannot make sure that the correct value of SK' is found. Thus, the attacker A is enforced to stop the protocol after message 4. One might argue that A may take her chance to generate a trial message 5 with the candidate sc and corresponding SK'. This can be tried, but the probability of success is the same as the probability of success of one trial of regular brute force attack, which is extremely small. Moreover, the attacker cannot try again with the same SK since after refusal of message 5 by the server, the protocol will start over with a different SK.

B. Replay Attacks

An adversary may choose to replay any previously sent message in the protocol runs. Replaying message 1 from an older session yields no result, since the server will most probably choose a different elliptic curve point and therefore will create a different SK. Similarly, replaying an older message 2 to client will still result in a different SK being created.

Since the client and the server choose generates different elliptic curve points at each run, SK will be different as well. Replay of message 3 will not be decrypted correctly at the server side and will not be of any advantage to the adversary since the server will reject due to incorrect ID_i. In the same sense, replaying messages 4, 5 or 6 will not benefit the attacker, because different SK at each iteration of the protocol leads to different SK'; therefore, the server will notice any discrepancy at the event of incorrect decryption.

C. Impersonation Attacks

Impersonation attacks happen when A successfully authenticates herself as either client or server to the other side.

1) Client Impersonation

A legitimate user requires a smart card, an ID and a password to log in. We do not focus on protecting the ID of the user, because users tend to select IDs that are not supposed to be private. The password and the smart card, however, should only be in possession of the user. In the situation that both the password and the card are compromised at the same time, A has everything required to authenticate as the real user. There is no way to prevent impersonation in this scenario, so we assume that A can only obtain one of these two, as discussed at the beginning of Section III.

If A knows the password, but does not have the smart card, then she cannot calculate SK' in the verification phase. She needs to know $H(b \,||\, PW_i)$ and for this, client's secret b should be known. However, this secret number is stored only in the smart card.

If A has the smart card, but does not have the password, she can still get past the login phase. Since the password space could be small, A may want to try all possible PW_i until successfully authenticated. This can be thwarted easily by employing a mechanism on the server side to lock down the IDs of users with a number of unsuccessful login attempts. Another approach A may want to take is to try and guess SK' then retrace from that point to find $H(b \,||\, PW_i)$. This attempt is futile since the message sent by the server (message 4) is highly entropic, because it is an encryption of a random number. The adversary may not know if he tries the correct key, since the decryption does not produce a "meaningful" result anyway.

2) Server Impersonation

In this section, we consider that the adversary A is a third party trying to impersonate server. Similar to the previous scenario, A can get past the login phase with ease. However, since H(b||PW) is unknown, A cannot resume the protocol by calculating SK′.

In another scenario, adversary may steal the card, extract b from the card, and place it back without the user gets noticed. In this case, when the user initiates the authentication process, the adversary may interrupt and try to impersonate the server. This attempt fails when client sends message 5. Since A cannot decrypt it since she does not know SK′, client will not get the response (message 6) and the protocol will fail.

IV. Implementation Details

We implemented the proposed protocol using a commercially available smart card (Feitian A22CR) over Java Card 2.2.2 platform. This section explains the details of this reference implementation.

Client side of the protocol runs on a smart card specified above. Server side of the protocol is managed by a Java application. This application handles server-side calculations as well as creation of commands that are to be sent to the smart card. In this way, the exchange of messages between the client and the server is simulated locally.

ECDH scheme is implemented by utilizing Java Card API's *KeyAgreement* class. The object from this class requires an *ECPublicKey* and an *ECPrivateKey* object to be initialized. As the mode of the key agreement scheme, *ALG_EC_SVDP_DH* is used.

Although there are a number of recommended elliptic curve parameters, smart cards only allow implementation of a limited subset of curves with specific prime lengths. In our implementation, an elliptic curve with domain parameters recommended under the name *secp192r1* [4] is constructed. To achieve this, keys are generated with *LENGTH_EC_FP_192* parameters.

For symmetric encryption, AES with block size of 128 bit is selected. The mode of the encryption is ECB and input data is not being padded. For hash operations, both SHA-1 and SHA-256 are available. One of them can be chosen at the registration phase that is used for the entirety of the protocol. For taking the timings, we use SHA-256.

To produce random challenge numbers, Java Card's $RandomData$ object is initialized with *ALG_SECURE_RANDOM* parameter. The random number generator in this mode produces cryptographically secure pseudo-random bit sequences.

The timestamps are not included in this implementation, since the platform needs an external resource to tell the time. This may be solved by implementing internal counters synchronized at the registration phase. The values read from these counters *may* be substituted with the timestamps.

V. Performance Evaluation

In this section, we provide performance evaluation of the proposed protocol. We first give the timings of our reference implementation and then we compare our protocol with that of two other protocols in the literature.

The computational timings of the protocol run are given Table 1 for the client side operations and in Table 2 for server side operations. The exact timings are given for each particular operation of login preparation and verification phases; timings of the enrollment phase are not considered since these operations are performed only once. The parameters used were given in Section IV. As the client side smart card, Feitian A22CR is used over Java Card 2.2.2. As the server, MacBook Pro, 2.5 GHz Intel Core i7 (OS X 10.11 El Capitan) is used. As seen in Table 1, the client side operations on the smart card takes almost 1.55 seconds, which is an acceptable delay for a smart card environment. After adding the server side delay, total computational delay of our proposed protocol becomes approximately 1.95 seconds. Such a delay is human-imperceptible for a login experience.

TABLE I. Client-side Timings

Process	Time (ms)
creation of message 1	522
secret key derivation, after msg. 2	753
creation of message 3	26
creation of message 5	152
session key calculation, after msg. 6	98
total	1551

TABLE II. Server-side Timings

Process	Time (ms)
creation of message 2	388
creation of message 4	15
creation of message 6	1
total	404

We also compared the computational cost of login and verification phases of our protocol to other schemes [3, 5, 6], which uses elliptic curve cryptography, in Table 3.

TABLE III. Comparison of Computational Cost

type of operation	Liu et al.[3]	Zhang et al.[6]	Chandrakar & Om[5]	Our Proposal
hash	14	10	4	6
EC point addition	2	4	2	0
EC point multiplication	3	14	7	4
symmetric en/decryption	1	4	4	8

This comparison is performed using the major cryptographic functions' counts. As seen in Table 3, our protocol uses much less number of EC point multiplications when compared two of other protocols [5, 6] and just one more EC point multiplication as compared to one of the protocols [3]. Multiplication of two elliptic curve points is the costliest operation among other operations such as symmetric encryption, decryption and hashing. Therefore, we conclude our proposed protocol performs better than two of the other works and comparable to one of them.

VI. Conclusion

We propose a two-factor remote authentication protocol that uses smart cards as what-you-have factor and a password as what-you-know factor. The proposed protocol is a robust one such that even if the smart card is captured and all the secrets stored in are revealed, the attacker cannot impersonate unless she know the password. Similarly, if the password is learnt by the attacker, she cannot launch any attack without possessing the smart card. More importantly, our protocol resists against smart offline dictionary attacks.

We implemented our protocol using a commercially available smart card and obtained reasonable timings, proving that the proposed protocol can be used practically. Moreover, we compared our protocol with other smart card based protocols in the literature and showed the superiority of the proposed one to two of the rival protocols.

References

[1] Kocher, P. C., Jaffe J., Jun B. 1999. Differential Power Analysis. Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag**:** 388-397.

[2] Messerges, T.S., Dabbish, E.A., Sloan, R.H., 2002. Examining smart-card security under the threat of power analysis attacks. IEEE Trans. Comput. 51 (5), 541–552.

[3] Liu, T. -H, H. -F Zhu, and J. -S Pan. 2012. "Robust and Efficient Password-Authenticated Key Agreement Scheme Based on Elliptic Curve Cryptosystem." Journal of Applied Sciences 30 (1): 67-74.

[4] Standards for Efficient Cryptography Group. 2010. "SEC 2: Recommended Elliptic Curves Domain Parameters, Jan. 2010. Version 2.0". http://www.secg.org/sec2-v2.pdf

[5] Chandrakar, P. and H. Om. 2015. "A Secure Two-Factor Mutual Authentication And Session Key Agreement Protocol Using Elliptic Curve Cryptography". *2015 IEEE International Conference On Computer Graphics, Vision And Information Security (CGVIS)*.

[6] Zhang, Y., Chen, J., Huang, B., & Peng, C. 2014. "An Efficient Password Authentication Scheme Using Smart Card Based on Elliptic Curve Cryptography," Information Technology And Control, Vol. 43, no.4, pp.390-401.

[7] Karuppiah, M., Saravanan R. 2014. "A Secure Remote User Mutual Authentication Scheme Using Smart Cards". Journal of Information Security and Applications. Volume 19, Issues 4–5, Pages 282–294.

[8] Song, R.. 2010. "Advanced Smart Card Based Password Authentication Protocol". Computer Standards & Interfaces 32 (5-6): 321-325. doi:10.1016/j.csi.2010.03.008.

[9] Xu, J., W.-T. Zhu, and D.-G. Feng. 2009. "An Improved Smart Card Based Password Authentication Scheme With Provable Security". Computer Standards & Interfaces 31 (4): 723-728.

[10] Islam, S. K.H. and G. P. Biswas (2013). "Design of improved password authentication and update scheme based on elliptic curve cryptography." Mathematical and Computer Modelling 57(11-12): 2703-2717.

[11] Li, X., Niu J., Khan M. K., Liao J. 2013. "An enhanced smart card based remote user password authentication scheme." Journal of Network and Computer Applications 36(5): 1365-1371.

[12] Wang, D., Wang P., Ma C., Chen Z. 2012. Robust smart card based password authentication scheme against smart card security breach. Cryptology ePrint Archive, Report 2012/439.