Keylogging-Resistant Visual Authentication Protocols

DaeHun Nyang, *Member, IEEE*, Aziz Mohaisen, *Member, IEEE*, and Jeonil Kang, *Member, IEEE*

Abstract—The design of secure authentication protocols is quite challenging, considering that various kinds of root kits reside in Personal Computers (PCs) to observe user's behavior and to make PCs untrusted devices. Involving human in authentication protocols, while promising, is not easy because of their limited capability of computation and memorization. Therefore, relying on users to enhance security necessarily degrades the usability. On the other hand, relaxing assumptions and rigorous security design to improve the user experience can lead to security breaches that can harm the users' trust. In this paper, we demonstrate how careful visualization design can enhance not only the security but also the usability of authentication. To that end, we propose two visual authentication protocols: one is a one-time-password protocol, and the other is a password-based authentication protocol. Through rigorous analysis, we verify that our protocols are immune to many of the challenging authentication attacks applicable in the literature. Furthermore, using an extensive case study on a prototype of our protocols, we highlight the potential of our approach for real-world deployment: we were able to achieve a high level of usability while satisfying stringent security requirements.

Index Terms—Authentication, smartphone, malicious code, keylogger

1 Introduction

Threats against electronic and financial services can be classified into two major classes: credential stealing and channel breaking attacks [20]. Credentials such as users' identifiers, passwords, and keys can be stolen by an attacker when they are poorly managed. For example, a poorly managed personal computer (PC) infected with a malicious software (malware) is an easy target for credential attackers [46], [51]. On the other hand, channel breaking attacks—which allow for eavesdropping on communication between users and a financial institution—are another form of exploitation [22]. While classical channel breaking attacks can be prevented by the proper usage of a security channel such as IPSec [13] and secure sockets layer (SSL) [43], recent channel breaking attacks are more challenging. Indeed, "keylogging" attacks—or those that utilize session hijacking, phishing and pharming, and visual fraudulence—cannot be addressed by simply enabling encryption.

Chief among this class of attacks are keyloggers [19], [44], [46]. A keylogger is a software designed to capture all of a user's keyboard strokes, and then make use of them to impersonate a user in financial transactions. For example, whenever a user types in her password in a bank's sign-in box, the keylogger intercepts the password. The threat of such keyloggers is pervasive and can be present both in personal computers and public kiosks; there are always cases where it is necessary to perform financial transactions using a public computer although the biggest concern is that a user's password is likely to be stolen in these computers. Even worse, keyloggers, often rootkitted, are hard to detect since they will not show up in the task manager process list.

To mitigate the keylogger attack, virtual or onscreen keyboards with random keyboard arrangements are widely used in practice. Both techniques, by rearranging alphabets randomly on the buttons, can frustrate simple keyloggers. Unfortunately, the keylogger, which has control over the entire PC, can easily capture every event and read the video buffer to create a mapping between the clicks and the new alphabet. Another mitigation technique is to use the keyboard hooking prevention technique by perturbing the keyboard interrupt vector table [42]. However, this technique is not universal and can interfere with the operating system and native drivers.

Considering that a keylogger sees users' keystrokes, this attack is quite similar to the shoulder-surfing attack. To prevent the shoulder-surfing attack, many graphical password schemes have been introduced in the literature [15], [18], [28]. However, the common theme among many of these schemes is their unusability: they are quite complicated for a person to utilize them. For some users, the usability is as important as the security, so they refuse to change their online transaction experience for higher security. The shoulder-surfing attack, however, is different from keylogging in the sense that it allows an attacker to see not only direct input to the computer but also every behavior a user makes such as touching some parts of screen. To adopt shoulder-surfing resistant schemes for prevention of keylogger is rather excess considering the usability. Notice that while defending against the shoulder-surfing attack is out of the scope of this work, and could be partly done using other techniques from the literature intended for this purpose, the promising future of smart glasses (like Google glasses)

- *D. Nyang and J. Kang are with the School of Computer and Information Engineering of Inha University, Incheon, Korea. E-mail: dreamx@seclab.inha.ac.kr.*
- *A. Mohaisen is with VeriSign Labs, 12061 Bluemont Way, Reston, VA 20190. E-mail: amohaisen@gmail.com.*

Manuscript received 9 Sept. 2013; revised 16 Jan. 2014; accepted 30 Jan. 2014. Date of publication 19 Feb. 2014; date of current version 26 Sept. 2014.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2014.2307331

makes the attack irrelevant to our protocols if it is to be implemented using them instead of mobile phones.

It is not enough to depend only on cryptographic techniques to prevent attacks which aim to deceive users' visual experience while residing in a PC. Even if all necessary information is securely delivered to a user's computer, the attacker residing on that user's computer can easily observe and alter the information and show valid-looking yet deceiving information. Human user's involvement in the security protocol is sometimes necessary to prevent this type of attacks but humans are not good at complicated calculations and do not have a sufficient memory to remember cryptographically-strong keys and signatures. Thus, usability is an important factor in designing a human-involving protocol [22].

Our approach to solving the problem is to introduce an intermediate device that bridges a human user and a terminal. Then, instead of the user directly invoking the regular authentication protocol, she invokes a more sophisticated but user-friendly protocol via the intermediate helping device. Every interaction between the user and an intermediate helping device is visualized using a Quick Response (QR) code. The goal is to keep user-experience the same as in legacy authentication methods as much as possible, while preventing keylogging attacks. Thus, in our protocols, a user does not need to memorize extra information except a traditional security token such as password or personal identification number (PIN), and unlike the prior literature that defends against should-surfing attacks by requiring complex computations and extensive inputs. More specifically, our approach visualizes the security process of authentication using a smartphone-aided augmented reality. The visual involvement of users in a security protocol boosts both the security of the protocol and is re-assuring to the user because she feels that she plays a role in the process. To securely implement visual security protocols, a smartphone with a camera is used. Instead of executing the entire security protocol on the personal computer, part of security protocol is moved to the smartphone. This visualization of some part of security protocols enhances security greatly and offers protection against hard-to-defend against attacks such as malware and keylogging attack, while not degrading the usability. However, we note that our goal is not securing the authentication process against the shoulder-surfing attacker who can see or compromise simultaneously both devices over the shoulder, but rather to make it hard for the adversary to launch the attack.

1.1 Scope and Contributions

In this paper, we demonstrate how visualization can enhance not only security but also usability by proposing two visual authentication protocols: one for password-based authentication, and the other for one-time-password (OTP). Through rigorous analysis, we show that our protocols are immune to many of the challenging attacks applicable to other protocols in the literature. Furthermore, using an extensive case study on a prototype of our protocols, we highlight the potential of our protocols in real-world deployment addressing users shortcomings and limitations. The original contributions of this paper are as follows:

- Two protocols for authentication that utilize visualization by means of augmented reality to provide both high security and high usability. We show that these protocols are secure under several real-world attacks including keyloggers. Both protocols offer advantages due to visualization both in terms of security and usability.
- Prototype implementations in the form of Android applications which demonstrate the usability of our protocols in real-world deployment settings.

We note that our protocols are generic and can be applied to many contexts of authentication. For example, a plausible scenario of deployment could be when considering the terminal in our system as an Automated Teller Machine (ATM), public PC, among others. Furthermore, our design does not require an explicit channel between the bank and the smartphone, which is desirable in some contexts; the smartphone can be replaced by any device with the needed functionality of capture photos (see Section 2 for more details). This property enables us to expand our visual authentication protocols into the service context using smart wearable devices, which will be mentioned in Section 4.

1.2 Organization

The remainder of this paper is organized as follows. In Section 2, we review the system, trust, and attacker models used in this paper. In Section 3, we present two novel authentication protocols. In Section 4, we extended the presentation of these protocols by discussing several implementation and design issues. In Section 5, we analyze the security of our protocols under several potential attacks. In Section 6, we report on several experiments and user studies to support the usability of our protocols. In Section 7, our protocols are assessed overall in terms of usability and security. In Section 8, we review related works from the literature. In Section 9, we draw concluding remarks and point out several future work directions.

2 System and Threat Model

2.1 System Model

Our system model consists of four different entities (or participants), which are a user, a smartphone, a user's terminal, and a server. The user is an ordinary human, limited by human's shortcomings, including limited capabilities of performing complex computations or remembering sophisticated cryptographic credentials, such as cryptographically strong keys. With a user's terminal such as a desktop computer or a laptop, the user can log in a server of a financial institution (bank) for financial transactions. Also, the user has a smartphone, the third system entity, which is equipped with a camera and stores a public key certificate of the server for digital signature verification. Finally, the server is the last system entity, which belongs to the financial institution and performs back-end operations by interacting with the user (terminal or smartphone) on behalf of the bank.

Assuming a smartphone entity in our system is not a far-fetched assumption, since most cell phones nowadays qualify (in terms of processing and imaging capabilities) to be the device used in our work. In our system, we assume that

there is no direct channel between the server and the smartphone. Also, we note that in most of the protocols proposed in this paper, a smartphone does not use the communication channel—unless otherwise is explicitly stated—so a smartphone can be replaced by any device with a camera and some proper processing power such as a digital camera, a portable music player with camera (iPod touch, or mobile gadget with the aforementioned capabilities) or a smart watch/glasses.

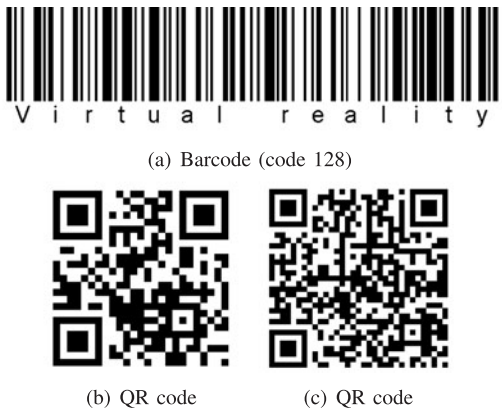
2.2 Trust and Attacker Models

For the trusted entities in our system, we assume the following: First, we assume that the channel between the server and the user's terminal is *secured* with an SSL connection, which is in fact a very realistic assumption in most electronic banking systems. Second, we assume that the server is secured by every means and is immune to every attack by the attacker; hence the attacker's concern is not breaking into the server but attacking the user. Finally, with respect to the keylogger attack, we assume that the keylogger always resides on the terminal. As for the attacker model, we assume a malicious attacker with high incentives of breaking the security of the system. The attacker is capable of doing any of the following:

- The attacker has a full control over the terminal. Thus,
  - While residing in a user's terminal, the attacker can capture user's credentials such as a password, a private key, and OTP token string.
  - The attacker can deceive a user by showing a genuine-looking page that actually transfers money to the attacker's account with the captured credentials that she obtained from the compromised terminal.
  - Or, just after a user successfully gets authenticated with a valid credential, the attacker can hijack the authenticated session.
- The attacker is capable of creating a fake server to launch phishing or pharming attacks.

For the smartphone in Protocol 1, we assume that it is always trusted and immune to compromise, which means no malware can be installed on it. Notice that this assumption is in line with other assumptions made on the smartphone's trustworthiness when used in similar protocols to those presented in this paper [34], [35], [41]. We, however, note that relaxing this assumption still could provide a certain level of security with Protocol 2. Protocol 2 uses two factors (password and the smartphone), and thus, the assumption can be relaxed so that not only the terminal but also smartphone could be compromised (one of them at a time but "not both together"). The non-simultaneous compromise assumption obviously excludes the shoulder-surfing attacker.

In our protocols, we also assume several cryptographic primitives. For example, in all protocols, we assume that a user has a pair of public/private keys used for message signing and verification. In Protocol 1, we assume that the server has the capability of generating one time pads, used for authentication. In Protocol 2, we assume users have passwords used for their authentication. Notice that these assumptions are not far-fetched as well, since most banking services use such cryptographic credentials. For example, with most banking services, the use of digital certificates issued by the bank is very common. Furthermore, the use of such cryptographic credentials and maintaining them on a smartphone does not require any technical background at the user side, and is suited for wide variety of users. Further details on these credentials and their use are explained along with the specific protocol where they are used in this paper.



(a) Barcode (code 128)

(b) QR code (c) QR code

Fig. 1. Three different barcodes encoding the statement "Virtual reality." (a) is a linear barcode (code 128), and (b) and (c) are matrix barcodes (of the QR code standard). While (b) encodes the plain text, (c) encodes an encrypted version using the AES-256 encryption algorithm in the cipher-block chaining (CBC) mode (note this last code requires a password for decryption).

2.3 Linear and Matrix Barcodes

A barcode is an optical machine-readable representation of data, and it is widely used in our daily life since it is attached to all types of products for identification. In a nutshell, barcodes are mainly two types: linear barcodes and matrix (or two dimensional, also known as 2D) barcodes. While linear barcodes—shown in Fig. 1a—have a limited capacity, which depends on the coding technique used that can range from 10 to 22 characters, 2D barcodes—shown in Figs. 1b and 1 c—have higher capacity, which can be more than 7,000 characters. For example, the QR code—a widely used 2D barcode—can hold 7,089 numeric, 4,296 alphanumeric, or 2,953 binary characters [4], making it a very good high-capacity candidate for storing plain and encrypted contents alike.

Both linear and matrix barcodes are popular and have been widely used in many industries including, but not limited to, automotive industries, manufacturing of electronic components, and bottling industries, among many others. Thanks to their greater capacity, matrix barcodes are even proactively used for advertisement so that a user who has a smartphone can easily scan them to get some detailed information about advertised products. This model of advertisement—and other venues of using these barcodes in areas that are in touch with users—created the need for barcode's scanners developed specifically for smartphones. Accordingly, this led to the creation of many popular commercial and free barcode scanners that are available for smartphones such as iPhone and Andriod phones alike.

3 Keylogging-Resistant Visual Authentication Protocols

In this section, we describe two protocols for user authentication with visualization. Before getting into the details of these protocols, we review the notations for algorithms used in our protocols as building blocks. Our system utilizes the following algorithms:

- $\mathsf{Encr}_k(\cdot)$. An encryption algorithm which takes a key k and a message M from set $\mathcal{M}$ and outputs a ciphertext C in the set $\mathcal{C}$.
- $\mathsf{Decr}_k(\cdot)$. A decryption algorithm which takes a ciphertext C in $\mathcal{C}$ and a key k, and outputs a plaintext (or message) M in the set $\mathcal{M}$.
- $\mathsf{Sign}(\cdot)$. A signature generation algorithm which takes a private key SK and a message M from the set $\mathcal{M}$, and outputs a signature σ.
- $\mathsf{Verf}(\cdot)$. A signature verification algorithm which takes a public key PK and a signed message (M, σ), and returns `valid` or `invalid`.
- $\mathsf{QREnc}(\cdot)$. A QR encoding algorithm which takes a string S in $\mathcal{S}$ and outputs a QR code.
- $\mathsf{QRDec}(\cdot)$. A QR decoding algorithm which takes a QR code and returns a string S in $\mathcal{S}$.

Any public key encryption scheme with indistinguishability against adaptive chosen ciphertext attacker (IND-CCA2) security would be good for our application. A public key encryption scheme with IND-CCA2 adds random padding to a plaintext, which makes the ciphertext different whenever encrypted, even though the plaintext is the same [26]. This restriction on the type of the used public key encryption scheme will prevent an attacker from checking whether his guess for the random layout is right or not. Thus, the security of the scheme is not dependent on the number of possible layouts but the used encryption scheme. If no such encryption is used, the adversary will be able to figure out the layouts used because he will be able to verify a brute-force attack by matching all possible plaintexts to the corresponding ciphertext. On the other hand, when such encryption is used, the 1-1 mapping of plaintext to cipher text does not hold anymore and launching the attack will not be possible at the first place. Also, any signature scheme with existential-unforgeability against adaptive chosen-message attacker (EUF-CMA) can be used to serve the purpose of our system. For details on both notions of security, see [16]. In particular, and for efficiency reasons, we recommend the short signature in [6].

3.1 Authentication with Random Strings

In this section, we introduce an authentication protocol with a one time password (OTP). The following protocol (referred to as Protocol 1 in the remainder of this paper) relies on a strong assumption; it makes use of a random string for authentication. The protocol works as follows:

1. The user connects to the server and sends her ID.
2. The server checks the ID to retrieve the user's public key (PK_{ID}) from the database. The server then picks a fresh random string OTP and encrypts it with the public key to obtain $E_{OTP} = \mathsf{Encr}_{PK_{\mathsf{ID}}}(OTP)$.
3. In the terminal, a QR code $QR_{E_{OTP}}$ is displayed prompting the user to type in the string.
4. The user decodes the QR code with $E_{OTP} = \mathsf{QRDec}(QR_{E_{OTP}})$. Because the random string is encrypted with user's public key (PK_{ID}), the user can read the OTP string only through her smartphone by $OTP = \mathsf{Decr}_k(E_{OTP})$ and type in the OTP in the terminal with a physical keyboard.
5. The server checks the result and if it matches what the server has sent earlier, the user is authenticated. Otherwise, the user is denied.

In this protocol, OTP is any combination of alphabets or numbers whose length is 4 or more depending on the security level required.

3.2 An Authentication Protocol with Password and Randomized Onscreen Keyboard

Our second protocol, which is referred to as Protocol 2 in the rest of this paper, uses a password shared between the server and the user, and a randomized keyboard. A high-level event-driven code describing the protocol is shown in Fig. 3. The protocol works as follows:

1. The user connects to the server and sends her ID.
2. The server checks the received ID to retrieve the user's public key (PK_{ID}) from the database. The server prepares π, a random permutation of a keyboard arrangement, and encrypts it with the public key to obtain $E_{KBD} = \mathsf{Encr}_{PK_{\mathsf{ID}}}(\pi)$. Then, it encodes the ciphertext with QR encoder to obtain $QR_{E_{KBD}} = \mathsf{QREnc}(E_{k_{\mathsf{ID}}}(\pi))$. The server sends the result with a blank keyboard.
3. In the user's terminal, a QR code ($QR_{E_{KBD}}$) is displayed together with a blank keyboard. Because the onscreen keyboard does not have any alphabet on it, the user cannot input her password. Now, the user executes her smartphone application which first decodes the QR code by applying $\mathsf{QRDec}(QR_{E_{KBD}})$ to get the ciphertext (E_{KBD}). The ciphertext is then decrypted by the smartphone application with the private key of the user to display the result ($\pi = \mathsf{Decr}_{SK_{ID}}(E_{KBD})$) on the smartphone's screen.
4. When the user sees the blank keyboard with the QR code through an application on the smartphone that has a private key, alphanumerics appear on the blank keyboard and the user can click the proper button for the password. The user types in her password on the terminal's screen while seeing the keyboard layout through the smartphone. The terminal does not know what the password is but only knows which buttons are clicked. Identities of the buttons clicked by the user are sent to the server by the terminal.
5. The server checks whether the password is correct or not by confirming if the correct buttons have been clicked.

4 Discussion

Some of the technical issues in the two protocols that we have introduced in the previous sections call for further discussion and clarification. In this section, we elaborate on how to

handle several issues related to our protocols, such as session hijacking, transaction verification, and securing transactions.

4.1 Password Hashing

Passwords are usually stored in a hashed form with a salt to prevent server attacks, instead of being stored in plaintext on the server. In Protocol 2, we can easily support this password hashing by making the server compare the password hash computed from the stored salt value and the transferred password after decrypting it with the stored password hash value.

4.2 Message Signing

For the generality of the purpose of this protocol and the following protocols, and to prevent the terminal from misrepresenting the contents generated by the server, one can establish the authenticity of the server and the contents generated by it by adding the following verification process. When the server sends the random permutation to the user, it signs the permutation using the server's private key and the resulting signature is encoded in a QR code. Before decrypting the contents, the user establishes the authenticity of the contents verifying the signature against the server's public key. Both steps are performed using the Sign and Verf algorithms. Verification is performed by the smartphone to avoid any man-in-the-middle attack by the terminal.

4.3 Prevention of Session Hijacking with Visual Signature Validation

Even with secure authentication, an attacker controlling entities in the system—the terminal in particular—via a malware can hijack the authentication session when a user tries to request some transactions such as money transfer. Though usually money transfer action prompts a user to input the password, the malware can easily hijack it and alter the transfer information with the attacker's information. To prevent the session hijacking in Protocol 1, we can make a QR code to include additional information on the user's transaction request as follows:

1. A user requests via terminal to the server money transfer denoted as T that describes sender name/account, recipient name/account, a timestamp, and amount of money to transfer.
2. The server checks the ID to retrieve the user's public key (PK_{ID}) from the database. Then, it picks a fresh OTP to prepare $QR = \mathsf{QREnc}(E_{OTP}, T, \sigma = \mathsf{Sign}(PrK, T))$, where PrK is a signing key of the server. Then, it sends QR to the user to authorize the transaction.
3. On the terminal, a QR code QR is displayed prompting the user to type in the OTP string.
4. The user decodes the QR code to get $(E_{OTP} = \mathsf{QRDec}(QR_{E_{OTP}}), T, \sigma)$ with her smartphone application. Here the application verifies the time stamp and the signature by $\mathsf{Verf}(PubK, T, \sigma)$ to show the result (Valid/Invalid) on the screen with the decrypted OTP and T. If the application fails to validate the signature, it does not show neither the decrypted OTP nor T, but displays an error message to alert the user. When the user is confirmed with the signature verification result and with T, she inputs the OTP to the terminal, which is sent back to the server.
5. The server checks the result and if it matches with the OTP that the server has sent earlier, the user is authenticated. Otherwise, the user is denied.

This expansion of the protocol enables a user to confirm that her critical transaction request has not been altered, and thus, the session hijacking attack is prevented. For this additional security functionality, a user's involvement is minimized in the protocol, because a user only sees the transaction information on the phone when it is valid, or an alert message when invalid. We note that the expansion is also applicable to Protocol 2, but not applicable to legacy OTP/password authentications.

4.4 Backward Visual Channel from PC to Smartphone

It is quite natural to think of the backward channel from PC to smartphone when PC has a camera. The idea of using QR code on the phone's screen as an upload channel (from phone to terminal) can be used, as previously done in other schemes [31]. Instead of using cellular network for a phone to send the confirmation, we can use this QR code-based visual channel from phone to a terminal.

The use of the visual channel to input encrypted credentials from the smartphone to the terminal has an interesting security implication. As is the case with using e-banking on untrusted terminals, imagine that such terminal is infected with a virus, or has a malware, which could be a keylogger. If the user is to use the authentication credentials directly on the terminal, it is obvious that these credentials will be compromised. On the other hand, if these credentials are keyed in on the smartphone and to be transferred using the visual channel between the smartphone and the terminal in an encrypted form for the server, the keylogging attacker will be prevented from logging these credentials on the terminal.

4.5 The Smart Glasses

The application context of our visual authentication protocols can be easily expanded by applying the visual authentication protocols to the smartphone and the glasses instead of the terminal and the smartphone. That is, an encrypted OTP for Protocol 1 or a blank keyboard for Protocol 2 appears on a smartphone instead of a terminal, and a user sees the secret information through the glasses instead of a smartphone to authenticate securely by entering the OTP or by tapping the password on the blank keyboard on the smartphone. Accordingly, the threat model must be redefined, and the usability study must be conducted again. The application-context change using the newly-arisen smart wearable devices such as smart glasses and smart watches together with a smartphone may lead to new security services supported by the visualization concept. We leave this for future work.

4.6 Time-Based OTP Devices

While one may consider independent OTP device such as RSA's SecureID and Google's Authenticator [1], [2], the way

both systems work prevent various advantages of our design if they are used in conjunction. Both of the SecurID and Google Authenticator are variations of the same idea, and they indeed implement the broader type of time-based authentication using RFC "6238: TOTP: Time-Based One-Time Password Algorithm" [37]. SecurID uses a tokenizer (at the client side; hardware or software) which has its own clock and a symmetric key, used as the seed for computing the token. At the other side, the server has a database with all legit "smart-cards" (tokenizers), a real clock, and keys. The server uses the key and the real clock value to re-compute the token generated by the user and validate the PIN sent by the user with a 60 seconds time-window. While SecurID is a proprietary software (closed source), Google authenticator is an open source, and software-based. It implements the token as the first few digits (6) of the HMAC-SHA1 of the current clock value on the device using the app. The same thing is done for verification as in SecurID. SecurID tokenizer refreshes tokens every 60 seconds and Google authenticator uses 30 seconds as a default.

While we use the concept of OTP, our use of tokens is in the form of challenge/response, differing from the time-based authentication schemes (represented by the two schemes above). Unlike both schemes, the OTP in our case is generated by the server, not the user, and is not timed (although the server may reject the once after a time-out by keep a counter of that OTP). For that reason, our system brings two advantages: (1) our system provides better mitigations to the replay attack. Everytime the user connects to the server, she's given a fresh token, whereas tokens in the time-based systems are refreshed every certain number of seconds. That window can be used by the attacker for launching a replay attack. (2) While mitigating the replay attack, our use of the challenge/response instead of time-based OTP is user friendly. It removes any stress on the users of having to type in the token within a short period of time.

4.7 Replacing Visual Channels with Bluetooth

The visual channel in Protocol 1 (that uses OTP) is used to transfer the encrypted OTP from PC to the smartphone, and the user plays a role of another channel from the smartphone to PC by entering the decrypted OTP into PC. Here, both channels (from PC to the smartphone and vice versa) can be replaced with other channels such as Bluetooth, and the whole authentication procedure can be automated. This will significantly enhance the usability of the authentication protocol. However, in another aspect, not all PCs are equipped with the Bluetooth module. Also, even though PC has the Bluetooth module, it might be an annoying job to execute the pairing whenever the user uses a device that she has never paired before. In that sense, Protocol 1 with visual channel and user's entering PIN are easier to be deployed in the current environment.

5 Security Analysis

Trust in our protocols can be seen shifted from PC to smartphone to make authentication protocols secure against malware in a PC. However, considering that it is not easy to protect user's credentials when a malware resides in a PC without sacrificing usability,[1] and that a user sometimes has to use an untrusted PC such as a public PC or a kiosk, our approach to move trust to the smartphone that is at least more trustworthy than public PCs is plausible. Also, in Protocol 2 that uses a password for authentication, a smartphone is not required to be trusted because a password is another factor for any successful authentication.

In this section, we analyze the security of our scheme under several attack scenarios and show how these attacks are defended against.

5.1 Key Space and Brute-Force Attacker

In our protocols, several stages include encryption of sensitive information such as credentials, which are of interest to the attacker (including the user ID, password, and nonce generated by the server). In our prototype, and system recommendations for wide use of our protocols, as well as the description provided above for the different protocols, we consider public key cryptography. Furthermore, we suggest a key length that provides good security guarantees. This includes the use of RSA-2048, which is infeasible to attack using the most efficient brute-force attack. This applies to both encryption and signature algorithms used in the protocols.

Notice that all public key cryptography in our protocols (except for signing and verification) can be replaced by symmetric key cryptography, which is far more efficient (despite that computation overhead in our protocols is marginal). Furthermore, such replacement of cryptographic techniques will not affect the security guarantees of our protocols if standard algorithms and key length are used—e.g., AES 128/256 [23], which is infeasible to brute-force. In our prototype, we use the latter symmetric key cryptography for securing communication.

5.2 Keyloggers

Keyloggers are popular and widely reported in many contexts [19], [21], [44], [46], [51]. In our protocols, input is expected by the user, and in every protocol one or another type of input is required. Our protocols—while designed with the limitations and shortcoming of users in mind, and aim at easing the authentication process by means of visualization—are aimed explicitly at defending against the keylogger attacks. Here, we further elaborate on the potential of using keyloggers as an attack, and the way they impact each of the two protocols.

Protocol 1. Authentication in this protocol is solely based on a random string generated by the server. The random string is encrypted by the public key of the user, and verified against her private key. The main objective of using OTP is that it is for one time use. Accordingly, if the keylogger is installed on the terminal, the attacker obviously will be able to know the OTP but will not be able to reuse it for future authentication. Alternatively, a keylogger installed on the smartphone will not be able to log any credentials,

1. It is very hard to detect rootkits with software when they reside in the OS's kernel, so nowadays dedicated hardware devices are being considered to detect rootkits [36]. Also, looking back at research on shoulder-surfing prevention, the shoulder-surfer resisting techniques are necessarily accompanied by degradation of usability somehow.

since no credentials are input on the smartphone. It is worth noting that the attacker may try to block users from being authenticated and reuse the OTP immediately. In this case, mitigations explained in Section 4.3 can be used to remedy the (session hijacking) attack.

Protocol 2. In the second protocol, a blank keyboard is posted on the terminal whereas a randomized keyboard with the alphanumerics on it is posted on the smartphone. Because the protocol does not require the user to do any keyboard input on the smartphone side, the protocol is immune against the keylogger attack. The user just checks the keyboard layout on the phone and there is no input from a user. Obviously, the terminal might be compromised, but the keylogger will be able to only capture what keystrokes are used on the blank keyboard. Thus, the keylogger will not be able to know which alphanumeric characters are being clicked.

5.3 Malicious Software

The term malware is generic, and is technically used to describe any type of code with malicious intentions including keyloggers. It is obvious that an attacker who successfully compromised a smartphone that has a private key that is a whole credential required to break the system in Protocol 1 will be always successful to break the systems except Protocol 2 that requires both password and private key.

5.4 Theft of Smartphone

In case the smartphone that has a key to recover OTP in Protocol 1 and to decipher the keyboard layout in Protocol 2 is stolen, obviously the security degrades. In Protocol 1, theft of smartphone means that the attacker has total control over user's account if the attacker knows the user's ID. Protocol 1 can be regarded as an authentication protocol requiring only one security token (a smartphone) and focusing on user convenience (the user does not need to memorize a password). However, in Protocol 2, it is not easy to sign in or to make valid transaction requests successfully because it requires not only the smartphone but also the password. Neither a password nor a password verifier such as a password hash is stored in the smartphone, so an attacker cannot mount the offline guessing attack. Protocol 2 is basically a two-factor authentication protocol that requires both a password and a smartphone, and thus, it is not vulnerable to theft of smartphone.

Even when a smartphone is susceptible to theft, a careful user can mitigate the impact of that event on the authentication protocols described in this paper and trust delegated to the smartphone. Many smartphones, like iPhone with iOS 5.0 or later, enable strong access control mechanisms that include a limited number of trials to input a non-simple passcode (an arbitrary alphanumeric string greater in length than four characters). Upon failure of inputting the correct passcode for more than the allowed number of times (default is eight times), the smartphone deletes all user contents, including applications as well. This, indeed, would allow the adversary only a very negligible capability in making use of credentials stored in the device for breaking the authentication mechanism proposed in this work even when the smartphone is stolen.

5.5 Shoulder-Surfing Attacks

As already mentioned in the introduction, shoulder-surfing resistance is not within our scope. However, in this section, we investigate the possibility and the effectiveness of shoulder-surfing attacks.

The shoulder surfing is a powerful attack in the context of password-based authentication and human identification [22], [30], [49]. In this attack, the attacker tries to know credentials, such as passwords or PINs by stealthily looking over the shoulder of a user inputting these credentials into the systems.

In Protocol 1, OTP tokens that have high entropy and are human-unfriendly making them hard to remember and recall are one-time used. Accordingly, a shoulder surfer would not benefit from launching an attack by trying to observe what the user at the terminal is inputting. The attack is not applicable to this protocol.

In Protocol 2, observing the terminal or the smartphone keyboard layout (on the smartphone screen) alone would not reveal the credentials of the user. Observing both at the same time in a shoulder surfing attack, and mapping stroked keys on the terminal to those on the smartphone screen would reveal the credentials of the user. Being able to successfully launch this attack is a non-trivial task, and requires the attacker to be in very near proximity to the user, which would raise the user's suspicions about the intentions of the attacker. However, because the attacker who successfully conducts all this necessary steps will get a password in Protocol 2, the protocol cannot be said to be secure against the shoulder-surfing attack. We leave it for the future work to make Protocol 2 secure against the shoulder-surfing attack by combining it with shoulder-surfing resistant schemes already explored in the literature [50].

Finally, while our design is not intended in its current form to defend against it, we note that the smart glasses such as the Google glass will easily frustrate the shoulder surfing attacker. In essence, this is because the keyboard layout will be shown only to the user wearing the glasses.

5.6 Comparison

To sum up, we compare the two protocols and the way they perform against several attacks. We consider the scenarios where the attacker has control over either the terminal or smartphone but not both of them at the same time. The comparison is in Table 1.

6 Implementation and User Study

In this section, we describe the details of the prototype implementations, and show the results of the user study for Protocol 2 using a numeric keyboard and using an alphanumeric keyboard. The numeric keyboard study was to know the speed and the error of the PIN entry, and the alphanumeric keyboard study was for the password entry.

6.1 Numeric Keyboard with Blank Space

We implemented Protocol 2 to see its usability for PIN, which is widely accepted for authenticating a person during banking transactions.

QR code in the protocol, which includes 164 characters at low error correction level, contains a JSON object [11] that

(a) QR Code Scanning (before) (b) QR Code Scanning (after) (c) Keyboard on Terminal

(d) Keyboard on Smartphone (e) Clicking Password on Blank Keyboard

Fig. 2. Photographs of the prototype we have developed to demonstrate our authentication protocols. (a) and (b) show the moments of a QR code scanning of a keyboard layout. (c) shows the blank keyboard shown at the terminal (on LCD screen). (d) shows the decoded randomized layout of the keyboard obtained from the QR code after decryption as viewed on smartphone. Note that the yellow square on which the mouse cursor is hovering in the terminal is shown through the smartphone to assist user's input. (e) shows that a user is clicking the password on the blank keyboard while seeing numbers through the smartphone.

consists of an encrypted keyboard layout, hashed user ID, and current time. The keyboard layout is encrypted by using AES-128 encryption algorithm with CBC mode and PKCS#7 padding. Base64 [24] was used for encoding the ciphertext and initial vector. We used ZBar android SDK 0.2 [8], an open source library for reading barcodes and QR codes. To speed up the reading QR codes, before running ZBar library, we let the application re-sample a capture image to a small image of which width is 500 pixel using nearest neighbor image scaling. After reading QR code, the smartphone displays a numeric keyboard on screen. The size of numeric keyboard is 4×4 and the numeric keyboard contains 10 numbers (0 to 9) in random positions. The rest of the positions remain in blank. Snapshots of our implementation are shown in Fig. 2.

```
01: user::user.send(server, id)
02: server::__upon_id_arrival:
03:     if(server.verify(id) == true):
04:         pkid = server.db.find(id)
05:         pi = server.generate_random_kb()
06:         ekbd = server.encrypt(pkid, pi)
07:         qrekbd = server.qrencode(ekbd)
08:         server.send(user, qrekbd)
09: terminal::__upon_qrekbd_arrival:
10:     terminal.view(qrekbd)
11:     terminal.view_blank_kb(pi)
12: smartphone::__upon_qrekbd_view:
13:     qrekbd = smartphone.capture(qrekbd)
14:     ekbd = smartphone.qrdecode(qrekbd)
15:     pi = smartphone.decrypt(skid, ekbd)
16:     smartphone.view(pi)
17: user::__upon_pi_view:
18:     pw = user.inputpassword(terminal)
19: terminal::upon_pw_input:
20:     terminal.send(server, pw)
21: server::__upon_pw_arrival:
22:     if(server.verify(id, pw) == true):
23:         server.authenticate(user)
24:     else:
25:         server.deny(user)
```

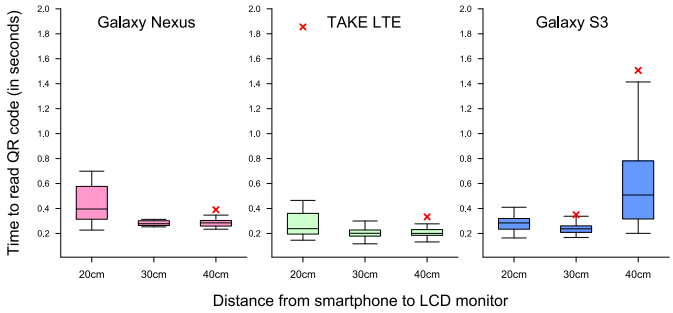
Fig. 3. High-level description of an authentication protocol with password and a randomized onscreen blank keyboard.

6.1.1 *Hardware Performance*

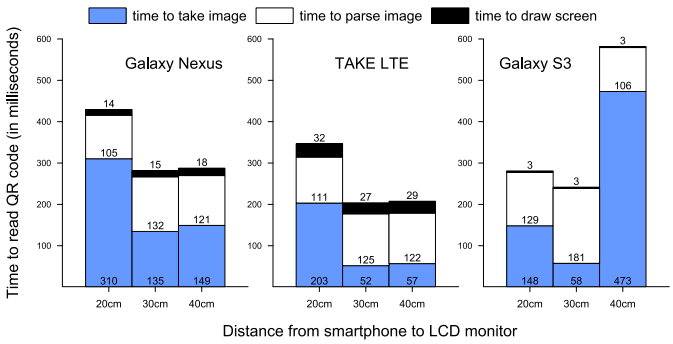
To understand how fast smartphones can read QR codes in our implementation, we measured the time to read QR codes by different distances from an LCD monitor (200 times each). QR codes are shown in $96 \times 96\text{mm}^2$ on LCD monitor, but the size of QR codes on smartphone varies according to the distance from the monitor or performance of a rear camera. Therefore, we performed the

TABLE 1
Comparison of Two Protocols and Their Resistance to Different Attacks When the Terminal and the Smartphone Are under Control of the Attacker

Attack	Brute-force	Keylogger	Malware
Protocol 1: OTP tokens			
Smartphone	✓	✓	
Terminal	✓	✓	✓
Protocol 2: Onscreen randomized keyboard			
Smartphone	✓	✓	✓
Terminal	✓	✓	✓



(a) times to read QR codes



(b) average times

Fig. 4. Time to read QR codes of three different smartphones by different distances from a smartphone to an LCD monitor. (a) shows the box plots with $1.5 \times \mathrm{IQR}$ whiskers and (b) shows average time for reading QR codes in each part: to take an image, to parse an image, and to draw screen. (size of QR codes on LCD monitor $= 96 \times 96\mathrm{mm}^2$, data length $= 164$ bytes)

experiments on three different smartphones: Galaxy Nexus, TAKE LTE, and Galaxy S3. Fig. 2 summarizes specifications of those smartphones.

As shown in Fig. 4, all smartphones can read QR codes very fast; in most cases, they can read QR codes in 300 ms. All smartphones show their best performance when they are located 30 cm far from the monitor. The average reading time (with 95 percent confidential interval) is $282.00(\pm 8.40)$ ms on Galaxy Nexus, $203.80(\pm 18.20)$ ms on TAKE LTE, and $241.60(\pm 21.16)$ ms on Galaxy S3 . However, they could not read well QR codes if the distance from LCD monitor is closer than 20 cm or further than 40 cm. A procedure to read a QR code can be divided into three parts: 1) taking an image, 2) parsing the image, and 3) re-drawing screen. In the first part, smartphones need to adjust their camera focus in order to take a QR code image. The time for the first part varies according to the distance from monitor and smartphones' camera performance. The second part is a sequential procedure to transform QR code image to keyboard layout. In a closer look, it includes tasks to re-sample the capture image (QR code), to transform the QR code to a JSON object, to parse the JSON object to a ciphertext, and to decrypt the ciphertext to a keyboard layout. To process the second part, it spent 122 ms on Galaxy Nexus, 120 ms on TAKE LTE, and 139 ms on Galaxy S3 on average. In the third part, smartphones need to re-draw the whole screen again in order to display the keyboard to user. Consequently, the time required to display the keyboard on screen is sufficiently short: it takes 15.72 ms on Galaxy Nexus, 29.21 ms on TAKE LTE, and 2.92 ms on Galaxy S3 on average of 200 trials.

TABLE 2
Specification of Smartphones Used in Experiments

	Galaxy U	Galaxy Nexus	TAKE LTE	Galaxy S3
By	Samsung	Google	KTTech	Samsung
SoC	S5PC111	OMAP 4460	MSM 8960	Exynos 4412
CPU	1GHz Coretex-A8	1.2GHz d Coretex-A9	1.5GHz d Coretex-A9	1.4GHz q Coretex-A9
GPU	-	PowerVR SGX54	Adreno 225	Mali-400 MP
Memory	512MB	1GB	1GB	2GB
Camera	5MP	5MP	5MP	8MP
Display	3.7″	4.65″	4.5″	4.8″
Android	2.3	4.1.2	4.1.2	4.0.4

The letters d and q with the CPU stand for dual and quad processors.

6.1.2 User Study Design

Protocol 2 with a numeric keyboard with blank spaces was evaluated using repeated measures within participants design of four-digit and eight-digit PIN. The purpose of the study was not to compare with the speed and the error of a control group such as a normal PIN entry method, but to know the speed and the error of our proposal, and therefore, we investigated parametric statistical values such as mean and standard deviation. This is because the keyboard size was 16 (for better security), and thus, a control group (the regular PIN pad has 10 keys only) could not be defined rigorously. In our experiment, a simulated server offered a randomized keyboard to the user at each authentication attempt, and the participants' mouse clicks were recorded with the time for later analysis.

6.1.3 Participants

We recruited participants for our study from our college. Each was given a coffee coupon. In total, 20 participants took part in the study. Among them, four participants were females, and 16 participants were males. They were 27.2 years old on average.

6.1.4 Procedure

In our user study for Protocol 2, we used a Galaxy Nexus specified in Table 2. Before the test session, we briefly explained to the participants how to authenticate with Protocol 2, and demonstrated them Protocol 2 with one of the authors' account. We did not give any training session, but they started the test session immediately after the demonstration. In the test session, they were asked to input PINs of their choices with two different lengths for registration: four and eight digits. Then, an authentication session started by prompting the participants to input their ID's. Just immediately after they entered ID, a QR code and a blank keyboard popped up as shown in Fig. 2, and participants read the QR code with an App in the smartphone. After reading the QR code, they were asked to input their PINs by clicking buttons on the blank keyboard of the terminal. When they reached the end of their PIN input, the click records were automatically submitted to the server (i.e., a participant was not asked to press a submission button). Each participant repeated this 10 times for four-digit and eight-digit PINs (in total, 20 times of authentication sessions). In the study, we found that many users knew they

TABLE 3
Results of the User Study for Protocol 2

$t(4)$	success	$t(8)$	success	gender (age)
10.4	●●●●●●●●●●	16.0	●○●●●●●●●●	male (26)
5.6	●●○●●●●●●●	8.3	●●●●●●●●●●	male (33)
8.5	●○●○●●●●●●	11.1	●●●●●●●●●●	male (22)
7.7	●●●●●●○●●●	7.4	●●●●●●●●●●	female (20)
11.6	○●●●●●●●●●	14.2	●●●●●●●●●●	male (27)
7.6	●●●●●●●●●●	9.9	●●●●●●●●●●	male (26)
7.0	●●●●●●●●●●	8.2	●●●●●●●●●●	male (28)
13.5	●●●●●●●●●●	17.0	●●●●●●●●●○	female (27)
5.0	●●●●●●●●●●	10.1	●●●●●●●●●●	male (29)
4.3	●●●●●●●○●●	7.9	●●●○●●●●○●	female (21)
9.7	●●●●●●●●●●	10.2	●●●●●●●●●●	male (43)
6.7	●●●●●●●●○○	8.6	●○●●●●●●●○	male (25)
4.1	●●●●●●●○●●	9.8	●●●●●●●●○●	male (25)
4.2	●●●●●●●●●○	14.4	●●●●●●●●●●	male (26)
8.5	●●●●●●●●●●	11.8	○●●●●●●●●●	male (25)
4.2	●●●●●●●●●●	8.9	●●●●●●●●●●	male (30)
6.9	●●●●●●●●●●	9.6	●●●●●○○●●●	male (30)
8.5	●●●●●●●●●●	11.5	●●●●●○●●●●	female (26)
7.0	●●●●●●●●●○	8.4	●●●●●●●●●●	male (23)
11.1	●●●●●●●●○●	13.6	●●●●●●●●●●	male (31)
7.6	94.5%	10.8	94.5%	4:16 (27.2)

$t(4)$ and $t(8)$ are the average time (in second) for each user (row) to input a PIN of the given length.

made wrong clicks before reaching to the last digit of a PIN, and they cancelled the session. We classified those trials as failure cases, of which the number can be seen in Table 3.

6.1.5 Results

Fig. 5 shows the empirical CDF of the time measurements for only successful trials and Table 3 shows detailed statistics and subjects. The mean (with 95 percent confidence interval), minimum, maximum, and median times it took (in seconds) were $7.647(\pm 0.457)$, 2.833, 18.753, and 7.047 with four-digit PIN and $10.850(\pm 0.457)$, 6.018, 26.132, and 9.924 with eight-digit PIN. In our experiment, the success rate is 94.5 percent for both four and eight digits PIN, which can be seen in Table 3. The study shows that Protocol 2 with PIN is sufficiently fast to be used.

6.2 Alphanumeric Keyboard

Alphanumeric passwords are widely used to sign into various types of servers, so we also developed a prototype of Protocol 2 with an alphanumeric keyboard as an Android application to see its usability. The application can run on any smartphone with Android OS [17] (version 2.2 or later). AES-192 encryption algorithm (in the counter mode) for contents encryption, Base64 encoding for byte-to-character encoding of encrypted contents, and uses ZXing [5], an open source implementation provided by Google for reading several standards of the 1D and 2D barcodes.

6.2.1 User Study Design

Protocol 2 with an alphanumeric keyboard was evaluated using repeated measures within participants design of the standard Qwerty keyboard, the randomized Qwerty (keys were shown on the terminal) and Protocol 2 (keys were shown through the smartphone). The purpose of the study was to compare with the speed and the error of Protocol 2 with the two control groups. The first case study with the standard Qwerty keyboard measured the response time of typical password input (without using our protocols). The second case study was the control group, where we use a randomized keyboard on the terminal, which was already

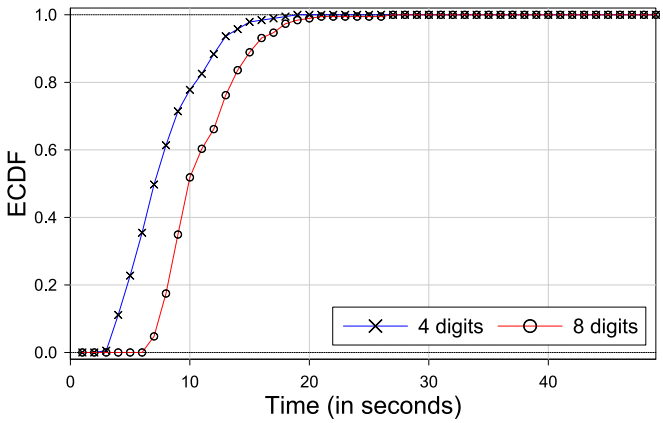


Fig. 5. Empirical CDF of the time it takes for inputting PINs of two different lengths in Protocol 2. The total numbers of successful trials are 189 (four digits) and 189 (eight digits).

used as a security mechanism (see Section 1). Accordingly, in the second case study, participants did not need to map the randomized keyboard to a layout on the terminal—they rather directly use the one on the terminal. The third case study was for Protocol 2, where participants used the smartphone to input their passwords. We used the same settings as in the prior two cases. In particular, the same sets of passwords with the previous control groups were used in this test. We measured the overall authentication time since the server introduced the randomized keyboard until the user logged in (or being rejected for password mismatch). In all cases, the users entered their passwords using mouse clicks. In the experiments, we used two password lengths four and eight characters as we did in Protocol 2 with PIN, where four-character passwords were used for investigating PIN-like environment and eight for typical non-simple passwords. Users chose passwords of their own. We repeated the study with each user 10 times to marginalize error and random effect in the experiments. In the study, we used a Samsung Galaxy U specified in Table 2.

6.2.2 Procedure

The procedure for this study was basically the same as that for the study with the numeric PIN. We asked 20 participants to input alphanumeric passwords of their own choice with different lengths (four and eight characters) on the Qwerty keyboard shown on the terminal, on the randomized keyboard and finally on the blank keyboard for Protocol 2. Because there were 36 keys (26 alphabets and 10 numbers) in Protocol 2, we printed on the blank keyboard the assisting numbers increasing from top-left to bottom-right, and on the smartphone, the randomized keys were displayed with the assisting numbers. Thus, it enabled participants to find easily the matched numbers once they found a key for their passwords. Input was done by mouse clicks. We measured the response time of each participant and repeated the test 10 times for each user and for four- and eight-character passwords (in total, 20 authentication sessions). We did the measurements for all cases, including when passwords were typed incorrectly.

In the following, we summarize the main results and findings from the case studies.

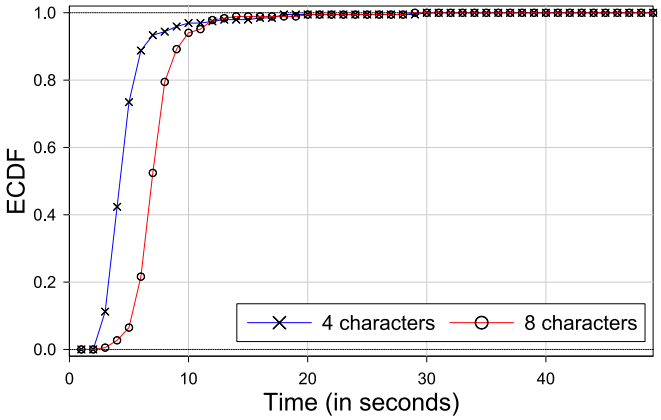


Fig. 6. Empirical CDF of the time it takes for inputting passwords of two different lengths. The total numbers of successful trials are 196 (four characters) and 185 (eight characters). The keyboard used for input is a qwerty keyboard, which is the typical Android phone keyboard.

6.2.3 Results - Normal QWERTY Keyboard

The average success rate was 98.0 percent with four-character passwords and 92.5 percent for eight-character passwords. An empirical CDF of the time measurements is shown in Fig. 6. We found that the mean, min, max and median (in seconds) were 4.25, 2, 29, and 4 when using four-character passwords and 6.74, 2, 28, and 6 when using eight characters.

6.2.4 Results—Random Alphanumeric Keyboard

*We found that the average success rate was 100 percent with four-character passwords and 94 percent with eight-character passwords. We plotted the empirical CDF of the time measurements in Fig. 7. We found that the mean, min, max, and median (in seconds) were 9.38, 3, 42, and 8 for length 4 and 14.77, 6, 46, and 14 for length eight passwords, respectively. We note that, on average, the time it took to input passwords using this method was twice as much as when using the qwerty keyboard.

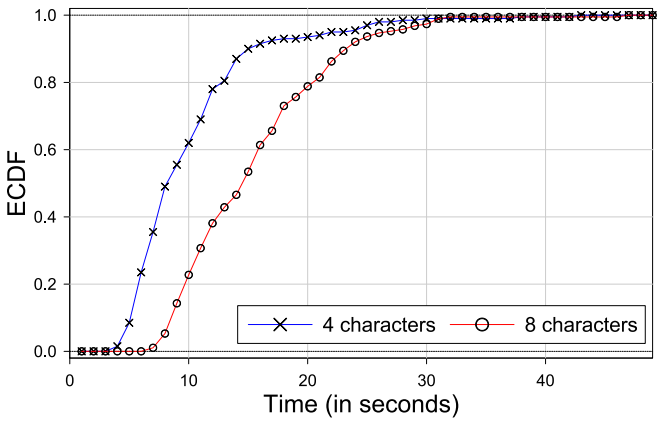


Fig. 7. Empirical CDF of the time it takes for inputting passwords of two different lengths. The total numbers of successful trials are 200 (four characters) and 189 (eight characters). The keyboard used for input is the a randomized Android phone keyboard (same keyboard is rendered on the smartphone and terminal).

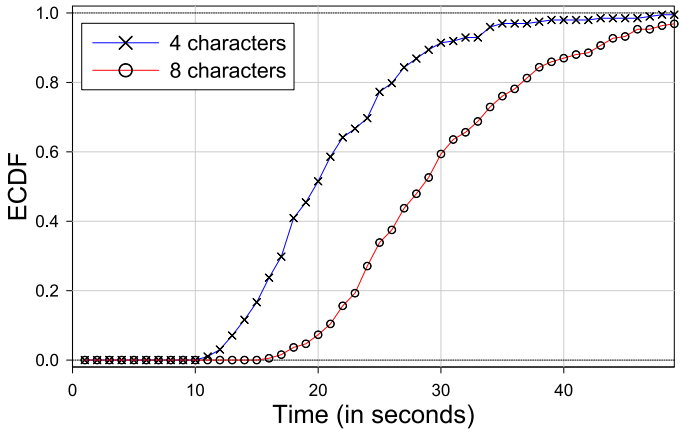


Fig. 8. Empirical CDF of the time it takes for inputting passwords of two different lengths in our protocol. The total numbers of successful trials are 198 (four characters) and 192 (eight characters).

6.2.5 Results—Protocol 2 with Randomized Alphanumeric Keyboard

We noticed that the average achieved success rate of our authentication protocol was 99.0 percent for four-character passwords and 96.0 percent for eight-character passwords. We notice that our system achieved a comparable success rate to that of both control groups supporting its high usability. Same as before, we plotted an empirical CDF of the time measurements in Fig. 8 (detailed statistics and subjects are in Table 4). The time measured in both figures includes the total time it took for cryptographic operations, encoding, decoding, communication (negligible), and user response. We notice that our system is practical compared to the other case studies, since the mean (with 95 percent confidential interval), minimum, maximum, and median times it took (in seconds) were 20.57($\pm$0.97), 10, 53, and 19 with four characters password and 29.44($\pm$1.18), 15, 62, and 28 with eight characters. Compared to the two other case studies, which do not provide the same security guarantees of our protocol, we find that our protocol takes on average

TABLE 4
Results of the User Study

$t(4)$	success	$t(8)$	success	gender (age)
25.2	●●●●●●●●●●	43.0	○●●●●●●●●●	male (32)
27.6	●●●●●●●●●●	37.6	●●●●●●●●●●	male (31)
21.1	●●●●●●●●●●	29.7	●●●●●●●●●●	male (29)
18.5	●●●●●●●●●●	24.9	●●●●●●●●●●	male (30)
25.3	●●●●●●●●●●	39.3	●●●●●●●●●●	male (29)
14.8	●●●●●●●●●●	23.4	●●●●○●●●●●	male (27)
16.9	●●●●●●●●●●	30.6	●●●●●●●●●●	male (28)
16.8	●●●●●●●●●●	23.6	●●●●●●●●●●	male (30)
13.1	●●●●●●●●●●	34.4	●●●●●●●●●●	male (24)
24.6	●●●●●●●●●●	39.9	●●●●●●●●●●	male (29)
28.1	●●●●●●●●●●	30.5	●●●●●●●●●●	male (27)
19.5	●●●●●●●●●●	24.9	●●●●●●●●●●	male (26)
23.0	●●●●●●●●●●	18.7	●●●●●●●●●●	male (25)
20.0	●●●●●●●●●●	26.5	●●●●●●●●●●	female (23)
24.1	●●●●●●●●●●	25.2	●●●●●●●●●●	male (29)
24.0	●●●●●●●●●●	27.3	●●●●●○●●○●	male (31)
18.1	○●●●●●●●●●	25.7	●●●●●●●●●●	male (29)
16.4	●●●○●●●●●●	35.1	●●○●●●○●●●	male (27)
16.7	●●●●●●●●●●	22.6	●●●●●●○●●○	male (30)
16.9	●●●●●●●●●●	25.9	●●●●●●●●●●	male (30)
20.6	99.0%	29.4	96.0%	1:19 (28.2)

$t(4)$ and $t(8)$ are the average time (in seconds) for each user (row) to input a password of the given length. The keyboard on the smartphone is re-randomized at each time.

TABLE 5
A Comparison with Other Works Based on Their Usability, Deployability, and Security, for Other Systems and How They Compare to Our Work, See the Work in [7]

	Usability								Deployability						Security										
Protocol	Memorywise-effortless	Scalable-for-users	Nothing-to-carry	Physically-effortless	Easy-to-learn	Efficient-to-use	Infrequent-errors	Easy-recovery-from-loss	Accessible	Negligible-cost-per-user	Server-compatible	Browser-compatible	Mature	Non-proprietary	Resilient-to-physical-observation	Resilient-to-targetted-impersonation	Resilient-to-throttled-guessing	Resilient-to-unthrottled-guessing	Resilient-to-Internal-observation	Resilient-to-leaks-from-other-verifiers	Resilient-to-phishing	Resilient-to-theft	No-trusted-third-party	Requiring-explicit-consent	Unlinkable
Password	◇	◇	●	◇	●	●	○	●	●	●	●	●	●	●	◇	○	◇	◇	◇	◇	◇	●	●	●	●
First	●	◇	○	◇	●	○	●	○	●	○	○	◇	○	●	●	●	●	●	●	◇	●	●	●	●	●
Second	◇	◇	○	◇	●	○	○	●	○	○	●	◇	○	●	●	●	●	●	○	◇	●	○	●	●	●
Phoolproof [41]	◇	◇	○	◇	●	○	○	◇	○	○	○	◇	◇	●	●	●	●	●	○	●	●	●	●	●	●
Cronto [3]	◇	◇	○	◇	●	○	○	◇	◇	○	◇	●	●	◇	●	●	●	●	○	●	●	●	●	●	●
MP-Auth [33]	◇	◇	○	◇	●	○	◇	○	○	○	◇	◇	◇	●	◇	○	◇	◇	◇	◇	●	●	●	●	●

Comparisons are in relation with password-based authentication, where ◇ stands for the case where the metric doesn't apply, ● stands for meeting the metric, ○ means that the metric can be made to work in the design, green and red are indicators for better and worse than the case of the password-based authentication, respectively.

twice as much time as the randomized keyboard method and roughly four times (about fives times for passwords with length 4) as much as the normal method (case study 1) for both passwords lengths. In relation with the time of other settings that do not consider any security assurances, our system shows reasonable usability features. However, compared to other systems that provide security guarantees, such as resisting password leakage attacks [50], our system is very highly usable and requires the same time overhead as in such systems.

Further details on the results of these case studies, and visualization of the response time of users are delegated to the Appendix.

7 Usability, Deployability, and Security: Related Work Perspective

Besides the security of an authentication protocol, both usability and deployability are equally important and critical for the acceptance of any protocol in modern computing settings. Bonneau et al. have developed 25 different metrics for evaluating such aspects in an authentication scheme to compete with the existing password-based authentication that is well-accepted in practice [7]. Furthermore, while those metrics are ideal, and the best authentication scheme in the literature does not address many of them, they are fairly generic to benchmark different designs and to compare them based on their merits. For that, the authors provided an extensive comparison and study of 38 schemes based on those metrics. Here, we benefit from this study in understanding our protocols in the context of the related works. We outline some of the merits based on the common features our schemes share with other works, and some others based on the prior user studies and security analyses we discussed in this paper.

The metrics are shown in Table 5. The reader is referred to [7] for further details on the definitions those metrics, and how they apply to the various authentication mechanisms in the literature. In the following, we summarize how our protocols perform on those metrics, and thus how they compare to other protocols in the literature. We limit our attention to the baseline, the password-based authentication, and a few phone-based authentication protocols as shown in Table 5. We notice that our first protocol meets the first metric by not requiring a password, meets the fifth, sixth, seventh, and eighth as shown in our user studies. Our design is security-rich, and its security features are discussed earlier to support the marked merits. Finally, for deployability, our system relies on an intensive user study that provides an obvious merit of its use against those metrics. The mapping of the metrics for the second protocols are also concluded from the prior discussion—details are omitted for the lack of space.

For the coloring part in the comparison, we use the same coloring system—with slightly different legend and keys—as utilized by Bonneau in his work, and the rationale of using those colors is explained therein. Our choice of PhoolProof and Cronto is not arbitrary: they share the same platform and some of the design aspects with our work, and thus it was easy to extrapolate many of the (subjective) coloring made by Bonneau to our work. This particularly explains much of the matching in deployability and security colorings for our protocol. The mismatch in "resilience to leaks from other verifiers" does not apply to our work, since it assumes a single verifier. The first protocol is more memorywise efficient, efficient to use, and infrequent errors than password based on the fact that it relies on OTP (not memorized by the user).

8 Related Work

There has been a large body of work on the problem of user authentication in general [27], [29], [38], [40], [45], and in the context of e-banking. Of special interest are

authentication protocols that use graphical passwords like those reported in [12], [18], [47], [48] and attacks on them reported in [10], [15], [18], [20], [25], [39]. To the best of our knowledge, our protocols are the first of their type to use visualization for improving security and usability of authentication protocols as per the way reported in this paper.

A closely related vein of research is trust establishment for group communication using cognitive capabilities. Examples of such works include SPATE [32], GAnGS [9], and SafeSlinger [14]. None of these works use visualization as reported in this work, although they provide primitives for authentication users and establishing trust.

Another closely related work is "Seeing-is-Believing" (SiB) [34] (extended in [35]), which uses visual channels of 2D barcodes to resist the man-in-the-middle attack in device pairing. Though we utilize similar tools by using the 2D barcodes for information representation, and the visual channel for communicating this information, our protocols are further more generic than those proposed in [34]. Our protocols are tailored to the problem settings in hand, e-banking, with a different trust and attack model than that used in [34]—which results into different guarantees as explained earlier in this paper. To prevent against phishing, Parno et al. suggested the use of trusted devices to perform mutual authentication and eliminate reliance on perfect user behavior [41]. Slightly touched upon in this paper are keyloggers as potential attacks for credentials stealing, which are reported in [19], [21], [44], and other malwares which are reported in [46], [51]. In this paper we have shown that our protocols are secure even when one of the participants in the authentication process (the terminal or smartphone) is compromised.

9 Conclusion and Future Directions

In this paper, we proposed and analyzed the use of user-driven visualization to improve security and user-friendliness of authentication protocols. Moreover, we have shown two realizations of protocols that not only improve the user experience but also resist challenging attacks, such as the keylogger and malware attacks. Our protocols utilize simple technologies available in most out-of-the-box smartphone devices. We developed Android application of a prototype of our protocol and demonstrate its feasibility and potential in real-world deployment and operational settings for user authentication. Our work indeed opens the door for several other directions that we would like to investigate as a future work. First of all, our plan is to implement our protocol on the smart glasses such as the google glass, and conduct the user study. Second, we plan to investigate the design of other protocols with more stringent performance requirements using the same tools provided in this work. In addition, we will study methods for improving the security and user experience by means of visualization in other contexts, but not limited to authentication such as visual decryption and visual signature verification. Finally, reporting on user studies that will benefit from a wide deployment and acceptance of our protocols would be a parallel future work to consider as well.

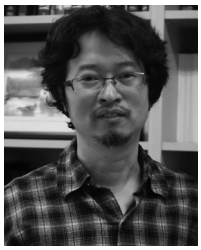
Acknowledgments

This work was supported by Inha University, Republic of Korea.

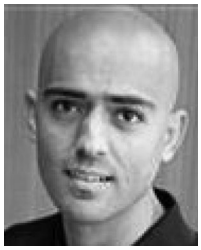
References

[1] Google Authenticator, http://code.google.com/p/google-authenticator/, 2014.
[2] RSA SecurID, http://www.emc.com/security/rsa-securid.htm, 2014.
[3] CRONTO, http://www.cronto.com/, 2014.
[4] *Information Technology. Automatic Identification and Data Capture Techniques*, BS ISO/IEC 18004:2006, ISO/IEC, 2006.
[5] ZXing, http://code.google.com/p/zxing/, 2011.
[6] D. Boneh and X. Boyen, "Short Signatures without Random Oracles," *Proc. Advances in Cryptology (EUROCRYPT)*, pp. 56-73, 2004.
[7] J. Bonneau, C. Herley, P.C. Van Oorschot, and F. Stajano, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," *Proc. IEEE Symp. Security and Privacy (SP)*, pp. 553-567, 2012.
[8] J. Brown, "ZBar Bar Code Reader, ZBar Android SDK 0.2," http://zbar.sourceforge.net/, Apr. 2012.
[9] C.-H.O. Chen, C.-W. Chen, C. Kuo, Y.-H. Lai, J.M. McCune, A. Studer, A. Perrig, B.-Y. Yang, and T.-C. Wu, "GAnGS: Gather, Authenticate'n Group Securely," *Proc. ACMMOBICOM*, pp. 92-103, 2008.
[10] S. Chiasson, P. van Oorschot, and R. Biddle, "Graphical Password Authentication Using Cued Click Points," *Proc. 12th European Symp. Research in Computer Security (ESORICS)*, 2008.
[11] D. Crockford, "The Application/JSON Media Type for Javascript Object Notation (JSON)," RFC 4627, http://www.ietf.org/rfc/rfc4627.txt, 2006.
[12] D. Davis, F. Monrose, and M. Reiter, "On User Choice in Graphical Password Schemes," *Proc. 13th Conf. USENIX Security Symp.*, 2004.
[13] N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 2003.
[14] M. Farb, M. Burman, G. Chandok, and J. McCune, "A. Perrig, "SafeSlinger: An Easy-to-Use and Secure Approach for Human Trust Establishment," Technical Report CMU-CyLab-11-021, Carnegie Mellon Univ., 2011.
[15] H. Gao, X. Guo, X. Chen, L. Wang, and X. Liu, "YAGP: Yet Another Graphical Password Strategy," *Proc. ACM Ann. Computer Security Applications Conf. (ACSAC)*, pp. 121-129, 2008.
[16] S. Goldwasser, S. Micali, and R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM J. Computing*, vol. 17, pp. 281-308, 1988.
[17] Google, "Android," http://www.android.com/, 2011.
[18] E. Hayashi, R. Dhamija, N. Christin, and A. Perrig, "Use Your Illusion: Secure Authentication Usable Anywhere," *Proc. ACM Fourth Symp. Usable Privacy and Security (SOUPS)*, 2008.
[19] C. Herley and D. Florencio, "How to Login from an Internet Café without Worrying about Keyloggers," *Proc. ACM Symp. Usable Privacy and Security (SOUPS)*, 2006.
[20] A. Hiltgen, T. Kramp, and T. Weigold, "Secure Internet Banking Authentication," *IEEE Security and Privacy*, vol. 4, no. 2, pp. 21-29, Mar./Apr. 2006.
[21] T. Holz, M. Engelberth, and F. Freiling, "Learning More about the Underground Economy: A Case-Study of Keyloggers and Dropzones," *Proc. 14th European Conf. Research in Computer Security (ESORICS)*, pp. 1-18, 2009.
[22] N. Hopper and M. Blum, "Secure Human Identification Protocols," *Proc. Advances in Cryptology (ASIACRYPT)*, 2001.
[23] R. Housley, "Using Advanced Encryption Standard (AES) Counter Mode with IPSec Encapsulating Security Payload (ESP)," RFC 3686, http://www.ietf.org/rfc/rfc3686.txt, 2004.
[24] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648, http://www.ietf.org/rfc/rfc4648.txt, 2006.
[25] C. Karlof, U. Shankar, J.D. Tygar, and D. Wagner, "Dynamic Pharming Attacks and Locked Same-Origin Policies," *Proc. 14th ACM Conf. Computer and Comm. Security (CCS)*, pp. 58-71, 2007.
[26] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC Press, 2008.

[27] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, http://www.ietf.org/rfc/rfc2104.txt, 1997.
[28] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, "Reducing Shoulder-Surfing by Using Gaze-Based Password Entry," *Proc. ACM Third Symp. Usable Privacy and Security (SOUPS)*, pp. 13-19, 2007.
[29] L. Lamport, "Password Authentication with Insecure Communication," *Comm. ACM*, vol. 24, no. 11, pp. 770-772, 1981.
[30] X.-Y. Li and S.-H. Teng, "Practical Human-Machine Identification over Insecure Channels," *J. Combinatorial Optimization*, vol. 3, pp. 347–361, 1999.
[31] J. Lim, "Defeat Spyware with Anti-Screen Capture Technology Using Visual Persistence," *Proc. ACM Third Symp. Usable Privacy and Security (SOUPS)*, pp. 147-148, 2007.
[32] Y.-H. Lin, A. Studer, Y.-H. Chen, H.-C. Hsiao, E.L.-H. Kuo, J.M. McCune, K.-H. Wang, M.N. Krohn, A. Perrig, B.-Y. Yang, H.-M. Sun, P.-L. Lin, and J. Lee, "SPATE: Small-Group PKI-Less Authenticated Trust Establishment," *IEEE Trans. Mobile Computing*, vol. 9, no. 12, pp. 1666-1681, Dec. 2010.
[33] M. Mannan and P.C. van Oorschot, "Leveraging Personal Devices for Stronger Password Authentication from Untrusted Computers," *J. Computer Security*, vol. 19, no. 4, pp. 703-750, 2011.
[34] J.M. McCune, A. Perrig, and M.K. Reiter, "Seeing-is-Believing: Using Camera Phones for Human-Verifiable Authentication," *Proc. IEEE Symp. Security and Privacy*, pp. 110-124, 2005.
[35] J.M. McCune, A. Perrig, and M.K. Reiter, "Seeing-is-Believing: Using Camera Phones for Human-Verifiable Authentication," *Int'l J. Security and Networks*, vol. 4, no. 1/2, pp. 43-56, 2009.
[36] H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek, and B.B. Kang, "Vigilare: Toward Snoop-Based Kernel Integrity Monitor," *Proc. ACM Conf. Computer and Comm. Security (CCS '12)*, pp. 28-37, 2012.
[37] D. MRaihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, http://www.ietf.org/rfc/rfc6238.txt, 2011.
[38] M. Naor and B. Pinkas, "Visual Authentication and Identification," *Proc. Advances in Cryptology (CRYPTO)*, 1997.
[39] M. Novoa, V. Ali, and M. Altendorf, *Virtual User Authentication System and Method*, US patent App. 20,080/028,441, 2006.
[40] D. Otway and O. Rees, "Efficient and Timely Mutual Authentication," *ACM SIGOPS Operating Systems Rev.*, vol. 21, no. 1, pp. 8-10, 1987.
[41] B. Parno, C. Kuo, and A. Perrig, "Phoolproof Phishing Prevention," *Proc. Financial Cryptography*, pp. 1-19, 2006.
[42] R. Pemmaraju, *Methods and Apparatus for Securing Keystrokes from Being Intercepted between the Keyboard and a Browser*, US patent 20070182714 A1, 2007.
[43] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.
[44] A. Slowinska and H. Bos, "Pointless Tainting?: Evaluating the Practicality of Pointer Tainting," *Proc. Fourth ACM European Conf. Computer Systems (EuroSys)*, pp. 61-74, 2009.
[45] J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An Authentication Service for Open Networks," *Proc. USENIX Ann. Technical Conf*, pp. 191-201, 1988.
[46] B. Stone Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your Botnet is My Botnet: Analysis of a Botnet Takeover," *Proc. ACM Conf. Computer and Comm. Security (CCS)*, pp. 635-647, 2009.
[47] X. Suo, Y. Zhu, and G. Owen, "Graphical Passwords: A Survey," *Proc. IEEE 21st Ann. Computer Security Applications Conf.*, 2005.
[48] J. Thorpe and P. van Oorschot, "Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords," *Proc. USENIX Security*, 2007.
[49] G. Vizcaino, *Method and Apparatus for Securing Credit Card Transactions*, US patent 5,317,636, 1994.
[50] Q. Yan, J. Han, Y. Li, J. Zhou, and R.H. Deng, "Designing Leakage-Resilient Password Entry on Touchscreen Mobile Devices," *Proc. Eighth ACM SIGSAC Symp. Information, Computer and Comm. Security (ASIACCS)*, pp. 37-48, 2013.
[51] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis," *Proc. ACM Conf. Computer and Comm. Security (CCS)*, 2007.



DaeHun Nyang received the BEng degree in electronic engineering from Korea Advanced Institute of Science and Technology, and the MS and PhD degrees in computer science from Yonsei University, Korea, in 1994, 1996, and 2000, respectively. He has been a senior member of the engineering staff at the Electronics and Telecommunications Research Institute, Korea, from 2000 to 2003. Since 2003, he has been an associate professor in the Computer Information Engineering Department at Inha University, Korea, where he is also the founding director of the Information Security Research Laboratory. He is a member of the board of directors and editorial board of the Korean Institute of Information Security and Cryptology. His research interests include cryptography and network security, privacy, usable security, biometrics and their applications to authentication and public key cryptography. He is a member of the IEEE.



Aziz Mohaisen received the MS and PhD degrees in computer science from the University of Minnesota, both in 2012. In 2012, he joined Verisign Labs, where he is currently a research scientist. Before pursuing graduate studies at Minnesota, he was a member of the engineering staff at the Electronics and Telecommunication Research Institute, a large research and development institute in South Korea. His research interests include the areas of networked systems, systems security, data privacy, and measurements. He is a member of the IEEE.

Jeonil Kang received the BS degree in computer engineering and the MS degree in information and telecommunication engineering both from INHA University, Korea, in 2003 and 2006, respectively, where he is currently working toward the PhD degree. His research interests include security issues of RFID, MANET, WSN, and biometrics. He is a member of the IEEE.