COMPUTERS & SECURITY ■■ (2017) ■■–■■

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

UFace: Your universal password that no one can see

Dan Lin [a,*], *Nicholas Hilbert* [a], *Christian Storer* [a], *Wei Jiang* [a], *Jianping Fan* [b]

[a] *Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409, USA*
[b] *Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223, USA*

ARTICLE INFO

Article history:
Available online

Keywords:
Face authentication
Privacy preservation
Android application
Secure multi-party computation
Impersonation attack

ABSTRACT

Due to the ease of use, face authentication could be a promising way to replace hard-to-remember passwords to access web services. However, to make face authentication suitable for web services, there are still several critical security and privacy challenges unaddressed. First, the existing authentication servers typically collect the plaintext of users' facial images in order to conduct authentication. If the servers are compromised, the attackers would obtain the users' facial images and can easily impersonate the users in any other applications that use face authentication. Second, it is also hard to prevent attackers from using facial images published in social network sites to impersonate the true user. In this paper, we conquered these two issues by proposing a novel secure face authentication system, called UFace. UFace uses special close-up facial images (which cannot be found online) for authentication. To further ensure the confidentiality of these close-up images, UFace guarantees that these images are only stored at user side and the servers have not any plaintext of these images. The face authentication is conducted securely with two collaborative authentication servers. UFace was implemented through both an Android application and multiple server side programs which were then evaluated in a real setting. The experimental results demonstrate that the UFace system can accurately authenticate users within a few seconds.

1. Introduction

With around 1 billion websites online today (Internet Live Stats, 2016), statistics (Tagat, 2012) show that each Internet user has an average of 26 different online accounts, with 25-to-34 year olds having an average of 40 accounts each. With so many different accounts that typically need passwords to access, some passwords are bound to be reused due to the challenge of memorizing many different passwords. The surprising fact is that a person usually uses just 5 unique passwords for all their accounts (Tagat, 2012). Using the same password across multiple accounts has opened the door to attackers and is becoming the main cause of the dramatic rise in online fraud. A common approach to reduce the user's burden on memorizing different passwords is password manager software. The password manager software can help generate, store and synchronize passwords among computers and personal devices of a single user. However, researchers (Li et al., 2014) have recently identified critical vulnerabilities in certain popular password manager softwares, e.g., LastPass, Password Box, etc., and revealed how an attacker could steal

* *Corresponding author.*
E-mail address: lindan@mst.edu (D. Lin).
https://doi.org/10.1016/j.cose.2017.09.016

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

(a) Close-up Photo (b) Zoom-In Photo

Fig. 1 – Close-up Photos vs. Zoom-In Photos.

arbitrary credentials from a user's account via these password manager software.

A question arises: Is there a way that does not require individuals to memorize many different passwords while still preventing attackers from accessing confidential information? Biometric authentication is one potential solution to this. Biometric authentication allows users to use fingerprints, iris, facial photos, etc. to authenticate themselves. In this work, we will focus on the face authentication. Being a newly emerging technique, face authentication needs to overcome several critical challenges, including: maintaining high accuracy of face authentication, and preventing impersonation attacks. With the advances in face recognition techniques that achieve over 90% of accuracy (Tan and Triggs, 2010), the accuracy is no longer a major hurdle. However, impersonation attacks have still not yet been fully conquered. Specifically, since many personal photos can be found online in social networks, attackers may directly use these online facial images to impersonate the true users. Although there have been liveness detection techniques (Li et al., 2015) by analyzing head movement, new way of attacks also emerge which could construct videos of head movement based on online images. Moreover, in existing face authentication systems, the users always need to disclose their facial images to the authentication server so that the server can verify the user's identity. If the server is compromised, an attacker or an insider can easily obtain the facial image of each person who is accessing the system, and then impersonate the user in other applications that use face authentication. These impersonation attacks could totally compromise the face authentication system.

In order to prevent the aforementioned attacks, we propose a novel face authentication system called UFace where "U" stands for "Your" as well as "Universal" to reflect our goal of creating a universal password based on each individual's facial images. UFace uses special close-up facial images for authentication. Specifically, when users needs to gain access to a web service, the user would take a close-up photo of his/her face using the UFace mobile application. Fig. 1 (a) is an example of a close-up photo. Such close-up images carry the personal device features, such as the camera's resolution and optical distortion caused by the short distance. More importantly, these close-up images are rarely shared online due to the lack of beauty (i.e., being distorted). Our experiments also show that these close-up images cannot be duplicated by attackers who try to use the same type of device to zoom in to take the victim's photo in a distance (such as the photo in Fig. 1 (b).

To further ensure the confidentiality of these close-up images, UFace guarantees that these images are only stored at user side and the servers have not any plaintext of these images. The face authentication is conducted securely with two collaborative authentication servers as shown in Fig. 2. In summary, our proposed UFace system has the following main technical contributions:

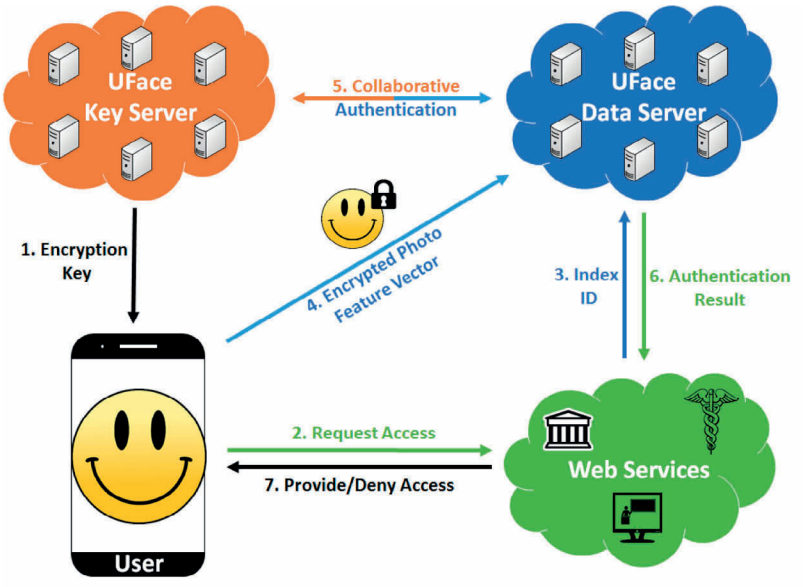


Fig. 2 – UFace System – Authentication Overview.

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

- The UFace authentication mechanism is unique in that the authentication servers do not have any plaintext of images while all the existing works require the servers to have a database of images. Specifically, the most related works in privacy-preserving face recognition (Erkin et al., 2009) do not need to see the user's plain text image but the authentication server still has a database of plaintext images for secure comparison. The existing work is not suitable in our case where all facial images need to be kept only at the user side so that users do not worry about their bioinformation being misused by others as passwords.
- We designed a novel homomorphic-based authentication protocol between the two UFace servers so that they are able to collaboratively conduct face image matching without actually looking at the plain text of the face images. Due to the complexity of the photo matching algorithms, the design of the collaborative homomorphic computation was very challenging. It required mapping and integration of various types of encrypted computations to work alongside garble circuit operations. The overall process also needed to be highly efficient so would provide negligible response time to end users.
- We developed an Android application that is capable of efficient photo feature extraction and encryption that keeps each users' computing burden to minimum. The development of the Android application involved multiple challenges that deal with the limited memory/computing power of mobile devices along with the need to design a new library for facial feature generation tailored to work on mobile device.
- We have evaluated UFace both theoretically and experimentally. The security analysis shows that UFace is robust against various types of attacks. The experimental results demonstrate that the UFace system can accurately authenticate users within a few seconds.

The rest of the paper is organized as follows. Section 2 discusses the related works on face authentication and face recognition. Section 3 introduces the background knowledge of several based techniques that form our proposed system. Section 4 gives an overview of the UFace system and the threat model. Then, Section 5 presents the UFace Android application at the user side and Section 6 presents the protocols at server side. Section 7 provides a security analysis of the system and Section 8 reports the performance study. Finally, Section 9 concludes the paper.

2. Related works

In this section, we discuss related works on privacy-preserving biometric authentication. Biometric authentication is very convenient for end users since it reduces the number of passwords to remember to zero. However, it also raises important privacy concerns since users' biometric data may be known by service providers or authentication servers (Bringer et al., 2013). One of the earliest attempts towards privacy preservation during biometric authentication is by Erkin et al. (2009). In their setup, the server has a set of photos that it does not want the user to see while the user has his/her own photo that needs to remain hidden from the server. They proposed a secure two-party comparison protocol that allows each user to check if his/her photo matches a photo in the server's database using Eigenfaces while keeping both the user's and the server's photos private to themselves. Later, Sadeghi et al. (2010) improved the efficiency of the above protocol. Following the similar settings, Osadchy et al. (2010) also proposed a privacy-preserving face detection algorithm – SCiFI – that allows a user to check if his/her photo is in the server's database without knowing the server's database. Evans et al. (2011) proposed a secure protocol for fingerprint matching while Blanton and Gasti (2011) proposed security protocols for both fingerprints and iris. Recently, Sedenka et al. (2015) employed a similar idea and implemented the privacy-preserving face authentication on smart phones. However, their system needs more than 10 minutes for a single authentication which is not suitable for real-time applications.

Compared with the aforementioned works, UFace has a totally different setting. The above works all assume that the authentication server has non-encrypted information, i.e., knows the unencrypted content of each user's biometric data. Unlike their works, the authentication servers in UFace only have access to encrypted feature vectors representing facial images. This setting significantly enhances privacy preservation and also introduces bigger challenges into the system design even though some of the same techniques are being utilized: garbled circuits and Paillier encryption.

Recently, there are several works which have similar security goals by having only encrypted data at the server side. One is by Blanton and Aliasgari (2012) who proposed both a single-server and a multi-server secure protocol to outsource computations of matching iris biometric data records. However, their single-server protocol uses predicate encryption scheme (Katz et al., 2008; Shen et al., 2009) which is not as secure as the additive homomorphic encryption scheme adopted into UFace. Their multi-server protocol leverages a secret sharing scheme (Shamir, 1979) and requires at least three independent servers, whereas our UFace system only needs two independent servers and is much more efficient. In Pal et al. (2015), authors proposed to watermark each user's facial image with fingerprints and then encrypt the watermarked biometric data to protect its privacy from adversaries. Their security protocol is conducted directly by the user and a single server, and hence the user bears a heavy computation workload. In UFace, the computation at the user side is lightweight, which helps conserve smart phone batteries. Another recent related work is by Chun et al. (2014) who developed a secure protocol that allows an organization to outsource encrypted users' biometric datasets to the cloud and let the cloud conduct authentication process on fully encrypted data. However, they mainly focus on fingerprint matching, the computation of which is much simpler than that for the face recognition on encrypted data in our system. Also, their algorithm takes over an hour to authenticate a user, which is not practical in a real world application.

In summary, there have been very limited efforts on privacy preserving face authentication and none of these existing work achieves the same security goal and efficiency as our proposed UFace system.

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

3. Background

In this section, we give a brief review of the fundamental techniques underlying our proposed system, including face recognition, Paillier cryptosystem, and garbled circuits.

3.1. *Face recognition*

Research on facial representation and recognition has been ongoing for numerous years. Two of the earlier methods for representing a person's face were Eigenfaces (Turk and Pentland, 1991) and Fisherfaces (Belhumeur et al., 1997). Later, a more advanced approach was proposed using so-called Local Binary Patterns (LBP) (Ahonen et al., 2004) to generate a feature vector for a photo. The most recent face recognition technique leverages deep convolutional neural networks (DCNN) (Luo, 2012; Sun et al., 2013) which typically use multiple layers of network for feature learning. However, DCNN based approaches are not applicable in our settings since our system does not allow the service provider to view the plain text of users' face images and hence the service provider would not be able to conduct the feature learning process. Therefore, we adopt the LBP based face recognition algorithm which already achieves a high accuracy rate under different environments (e.g., different lightings). In what follows, we briefly review the LBP-based face recognition approach.

The original LBP method follows a straightforward algorithm of picking an individual pixel and comparing its intensity against the 8 surrounding pixels' intensity (intensity is used since every image is first converted to gray-scale). If the surrounding pixels' intensity was greater than or equal to the intensity of the center pixel then it would be represented by a 1, otherwise it was given a 0. From this point, the 8 surrounding pixels are given a bit of information so the collection of these pixels is a byte of information which is called a label in LBP terms. This label is generated from starting at the pixel above and to the left of the center pixel and then reading each bit in a clockwise pattern.

An example of the basic LBP operation is shown in Fig. 3 where a pixel with an intensity value of 92 was given a label of 01010000. This process is repeated for every pixel in the image to generate a histogram.

The value for each bin of the histogram is the number occurrences of the specific encoding in the facial image. Since there are 8 bits used to encode a single pixel, there are $2^8 = 256$ possible labels. This means the histogram will be a vector of length 256; however, this can be reduced by using something called uniform labels. A label is considered uniform if there are at most two bitwise transitions in the encoding (ie. a change from 0 to 1 or vice versa). The label 01010000 would not be uniform since there are 4 transitions, while 00111000 would be uniform since there are only 2 transitions. All non-uniform labels can be placed into a single seperate bin. Thus, since there are 58 uniform values between 0 and 2^8 and 1 bin for all non-uniform values, the histogram is reduced to only needing n = 59 bins.

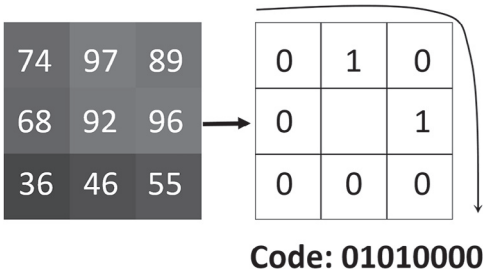


Fig. 3 – An example of computing the LBP for a pixel.

This current LBP scheme does not maintain spatial relation, however. This can be fixed by dividing the image into separate regions and calculating the histogram for each region. This allows for more efficient label comparison since pixels' labels have a smaller domain of other pixels to match with.

For example, if an image is 256 by 256 pixels and is separated into $k = 16$ equal sized sections, then each section of the image will contain $64 \times 64 = 4096$ pixels. LBP is then done in each of the 16 sections to obtain 16 different histograms. These histograms are then concatenated together in the form $\{H_1H_2 \ldots H_k\}$ to form the feature vector of the face which would be $16 \times 59 = 944$ bins. It should also be noted that the max value in any bin is equal to the number of pixels in a section. Since there are 4096 pixels in each section, the max value in any bin is 4096 which can be respresented with 13 bits (ie. $2^{12} + 1 = 4096$).

To compare two feature vectors of faces, standard histogram comparison techniques can be used such as Histogram Intersection. Given two histograms A and B with n bins, the intersection is defined as:

$$\sum_{i=1}^{n} \min(A_i, B_i)$$

This formula can be normalized to:

$$H(A,B) = \frac{\sum_{i=1}^{n} \min(A_i, B_i)}{\sum_{i=1}^{n} B_i}$$

where $H(A, B)$ is a percentage showing the closeness of to histograms. This is easily converted to be used with LBP with the following modification:

$$\frac{\sum_{j=1}^{k}\sum_{i=1}^{n} \min(A_{ji}, B_{ji})}{\sum_{j=1}^{k}\sum_{i=1}^{n} B_{ji}}$$

where j is the region index. It should be noted that if histograms are concatenated together, they behave like a single giant histogram for the purposes of histogram intersection.

3.2. *Paillier cryptosystem*

This type of cryptosystem is known as an additive homomorphic public-key encryption scheme. In public-key cryptosystems, a public key is used to encrypt a piece of information and a separate private key is used for decryption. In this setting, an authenticator generates both keys and distributes the public key while keeping the private key secure. Then, when a message needs to be sent to the authenticator, it is first encrypted using

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

the public key and then decrypted once it reaches the destination.

There are 2 unique properties of Paillier's encryption scheme. The first is that it is an additive homomorphic scheme. This means that it is possible to compute the encrypted sum of encrypted messages ($E(m_1)\cdot E(m_2)\equiv E(m_1+m_2)$) and the encrypted multiplication of encrypted messages ($E(m)^k\equiv E(k\times m)$). This property allows for operations to be computed securely on an already encrypted message without needing to decrypt the message first. The second property is that it is semantically secure which guarantees that a ciphertext will reveal no information about the plaintext. The reason this is ensured, is that for every encryption, a random value is introduced into the encryption. This means that the same message encrypted multiple times will output different ciphertexts. For more thorough details on Paillier's encryption scheme, see Paillier (1999).

3.3. *Garbled circuits*

The goal of garbled circuits is to provide a secure computation for multiple parties to compute a function in which no party learns the inputs of any other party. A circuit can be considered to be a sequence of boolean gates which are able to compute a specific function. Once the circuit is generated, the inputs are obfuscated with random keys for each input wire of the circuit. The garbled circuit is then sent to the second party where they obtain their inputs to the circuit through oblivious transfer and can evaluate the circuit securely. Since each wire is obfuscated, no party can learn anything about the inputs of any other party. For more thorough details on garbled circuits, see Yao (1986).

4. System overview

In this section, we first present an overview of the UFace system. Then, we introduce the threat model and security goals.

4.1. *System overview*

The UFace system is targeting smart phone users with the aim to assist them in accessing online services easily, safely and privately.

We design a privacy preserving face authentication framework called UFace that prevents the web service providers and authentication servers from seeing the actual facial images used by the users for authentication. The UFace system serves as a middle man between multiple web service providers and users. The framework of UFace is shown in Fig. 2 which involves three types of entities: users, web service providers and UFace authentication servers. The UFace authentication servers can be further classified into two types: (i) UFace data server; and (ii) UFace key server. The data server is in charge of storing encrypted user data while the key server manages keys. The two servers need to execute secure collaborative protocols during the authentication process. This design follows the spirit of "separation of duty" to achieve privacy preservation. It is worth noting that our UFace framework can be extended to support multiple data servers and multiple key servers to further enhance the efficiency. In the following discussion, we focus on two servers for easy illustration of the main ideas.

The UFace authentication process consists of two main phases: (i) registration phase; and (ii) authentication phase.

- **Registration phase**: When a user registers with a web service provider that uses UFace for authentication, the user will be directed to the client-side UFace application and asked to take a few selfies in short distance. The UFace application will extract feature vectors of these close-up photos, encrypt the feature vectors, and send them to the UFace data server. The UFace data server will register the user's device and evaluate the quality of the received selfies by collaboratively examining their feature vectors with the UFace key server. Note that none of the UFace servers will see the actual photos. All operations are conducted directly on encrypted photos. Once the selfies pass the evaluation, the UFace data server will store them for the subsequent authentication.
- **Authentication phase**: When a registered user wants to access the web service, he first takes a photo of himself using the camera on his mobile phone. The UFace app will send the user's web ID to the web service provider and his encrypted photo feature vector to the UFace data server. In order to keep users' close-up images confidential, the UFace data server does not have the key to decrypt the photo features. The data server will conduct a secure multi-party computation protocol with the UFace key server to collaboratively figure out whether the newly submitted photo matches the stored photo. The secure multi-party computation protocol ensures that information at each server remains confidential to each server. Therefore, even though the key server possesses the decryption key, the key server will not see the actual photo during the entire authentication protocol. The final authentication result will be forwarded to the web service provider who will then grant access to the user accordingly.

4.2. *Threat model and security goals*

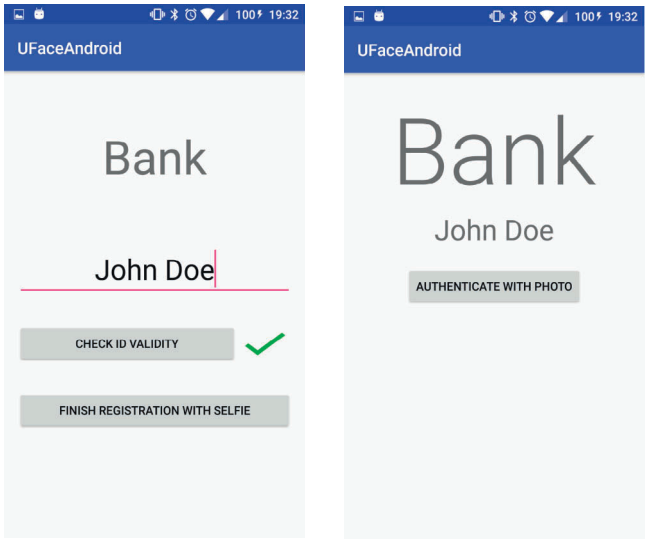
In UFace system, we adopt the commonly used semi-honest security model which assumes that each participating party will follow the protocols but they may be curious and try to explore the information available to them (Goldreich, 2004). In general, secure protocols under the semi-honest model are more efficient than those under the malicious adversary model, and almost all practical SMC protocols proposed in the literature (Ben-David et al., 2008; Canetti, 2000; Goldreich, 2004; Katz and Lindell, 2007) are secure under the semi-honest model. Under the semi-honest model, the participating parties would not collude. In our case, it means that the two authentication servers, i.e., the data server and the key server, do not collude. This can be guaranteed by deploying the two servers in two different clouds such as Amazon and Microsoft whereby the two big cloud service providers have no incentive to collude.

The security goal of our UFace system is to keep users' authentication information fully private, which includes the following aspects:

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

- Users do not need to reveal the actual content of their facial images (used for authentication) to any party during the authentication process.
- Web service providers can safely outsource the authentication process to the UFace authentication servers without violating users' privacy concerns regarding their facial images that have been used for authentication.
- UFace authentication servers do not have any plaintext of users' facial images. That means UFace servers do not possess users' bioinformation.
- UFace authentication servers can not connect any encrypted information back to any specific user.

In the following sections, we will present the UFace application at the client side and the privacy preserving protocols at the server side respectively.



(a) Registration (b) Authentication

Fig. 4 – Snapshots of UFace App.

5. UFace Android application

The UFace Android application consists of two modules: *Web Service Access* and *UFace Pass Generation*. Users will directly interact with the web service access module to log in to the web services. The UFace Pass generation module is executed automatically in the background to generate encrypted feature vectors based on the user's selfies during the authentication process. Each module is elaborated in the following.

5.1. *Web service access*

After users installed the UFace app, the first thing that they need to do is to add web services to their apps. Specifically, when the UFace app is launched, a blank screen will be shown with a "+" icon. By clicking on the "+" icon, a new window will open to show a list of web services which use the UFace system for authentication (obtained from UFace data server automatically). Once the user selects a web service, a registration page will appear. On the registration page, the user will see a regular log-in page to create a unique user ID and "UFace Pass" for the web service. The user ID will be sent to the web service provider to verify the uniqueness (as shown in Fig. 4(a)). If the user ID is good to use, the web service provider will return a so-called "Index ID" to the user. This "Index ID" indicates the location where the user's UFace Pass will be stored at the data server and also prevents the data server from knowing the user's actual user ID. By creating different Index IDs, it would be easy to extend current system to accommodate multiple user devices registered for the same web service.

The "UFace Pass" is actually the encrypted feature vector of the user's selfie and additional system parameters, which will be used as the password for the user to access this web service later on. To create a UFace Pass, the app will ask the user to take a couple of close-up selfies and then the app will automatically generate the UFace Pass for the user (the generation algorithm is presented in Section 5.2). To preserve confidentiality, the app will discard the selfies immediately after the generation of the UFace Pass, and then discard the UFace Pass after it has been transferred to UFace authentication servers.

Later, when a user needs to access a web service via UFace, he will see a list of web services that he has added on the launch screen of the app. By clicking on the desired web service, the user will be directed to a log in page where he will be asked to input the user ID for that service and to take a close-up selfie (Fig. 4(b)). The UFace app will send the user ID to the web service provider while the Index ID and the UFace Pass to the UFace data server. If access is granted, the user will be directed to his account in the web service's website.

5.2. *UFace Pass generation*

The overall authentication process is shown in Fig. 5. The most important feature of the UFace app is the UFace Pass generation which is responsible for converting the user's selfie into the encrypted feature vector and send it to the authentication server. There are two main steps: (i) feature vector generation and (ii) feature vector encryption.

5.2.1. *Feature vector generation*

When the user is asked to take a selfie, the selfie needs to be a close-up image that fills the whole screen of the smart phone as shown in Fig. 1(a). This close-up image is stored as a bitmap which allows for easy bit manipulation and thus easy execution of the LBP algorithm. As introduced in Section 3.1, an LBP photo feature vector can be represented in the form given by Definition 1.

Definition 1. (Feature Vector). Let p be a photo. Its feature vector F_p is represented as $F_p = \langle \overrightarrow{v_1}, \ldots, \overrightarrow{v_m} \rangle$, where $\overrightarrow{v_i} = \langle b_1, \ldots, b_{59} \rangle$ $(1 \le i \le m,\ m = 2^{2n})$.

Since a photo has pixels equal to a power of 2, partitioning the grid into 2^{2n} sections will yield equal-sized squares for

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

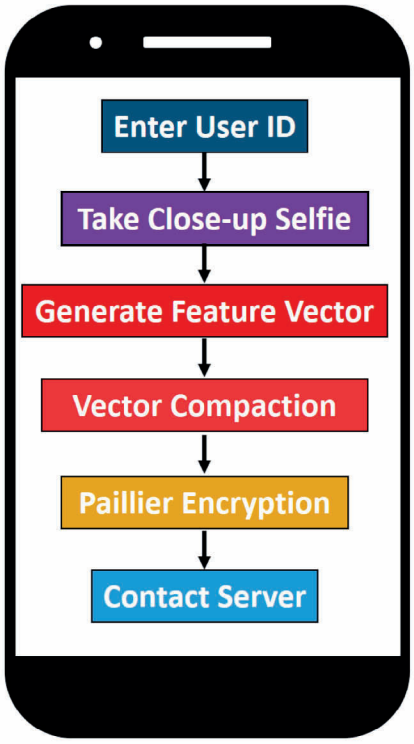


Fig. 5 – Authentication Process at Client Side.

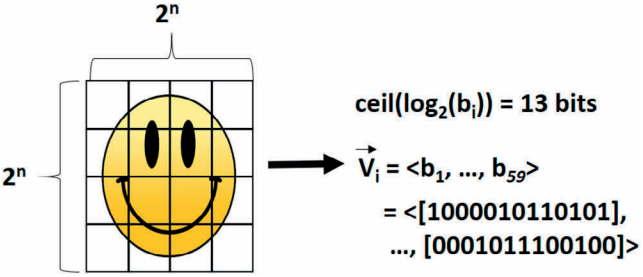


Fig. 6 – Feature vector generation.

easy comparison in the subsequent steps. As shown in Fig. 6, photo features in each square are represented using a vector that contains 59 bins based on the LBP algorithm.

5.2.2. *Feature vector encryption*

After obtaining the feature vector, the next step is to encrypt it using the public key of the UFace key server (which is obtained each time the application is started). Encryption is done using the Paillier's encryption scheme (Paillier, 1999). Paillier is an additive homomorphic and probabilistic asymmetric encryption scheme. The additive homomorphic property is described as follows: Let $[a]$ and $[b]$ be the encrypted form of a and b respectively. Without decrypting $[a]$ and $[b]$, $[a+b]$ can be computed by $[a] \times [b]$. In other words, the multiplication of two encrypted values yields the encryption of the sum of the

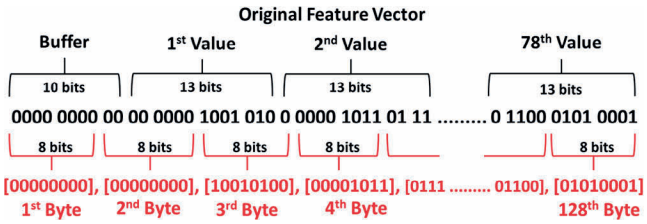


Fig. 7 – Feature vector compaction.

two original values. The encryption scheme is also semantically secure (Goldwasser and Micali, 1984), i.e., given a set of ciphertexts, an adversary cannot deduce any information about the plaintext.

The encryption of the feature vector needs to be very efficient as the user is trying to accomplish the encryption on a smart phone and in real-time. Thus, encrypting every value of each bin of each square of the photo would be too expensive and take too long. For example, for a photo divided into 16 sections, there would be $16 \times 59 = 944$ different bins and hence 944 individual encryptions with the feature vector of size $944 \times 2048 = 1{,}933{,}312$ bits (i.e., 236KB). To make the encryption more efficient, we propose the following approach.

The main idea is to concatenate the bits in consecutive bins into single numbers which have the bit size equivalent to the encryption key size as shown in Fig. 7. Specifically, we first compute the maximum value that a bin can have, which is equivalent to the total number of pixels in each square section of the grid. Given a photo with total 2^x pixels, the number of pixels in each square section of the grid will be 2^{x-2n}. If all pixels in the section have feature values falling into a single bin, the bin will need to be represented by $x - 2n + 1$ bits. A simple example is that if $n = 2$ and $x = 16$, then $2^{x-2n} = 2^{12}$ or 4096 pixels per section (0 to 4095). If all those values are placed into a single bin, the count will become 4096 which needs to be represented using $13 = 12 + 1$ bits. This is why there is the "+1" in the above calculation.

Using the previous example and having the 1024-bit encryption key, we can represent $\lfloor 1024/13 \rfloor = 78$ bins in a single value. There would be 10 bits left over. We set the first 10 bits of the 1024-bit value to "0" to avoid having negative values that could impact the subsequent photo matching computation. Given total 944 bins for a photo, only $\lceil 944/78 \rceil = 13$ encryptions are needed. This is a great increase to computation speed because instead of encrypting 944 numbers and transmitting 236KB data, our approach only encrypt 13 numbers and transmitting 3KB data.

After the encryption of the feature vector is completed, the encrypted feature vector along with the Index ID will be sent to the data server for either registration or authentication. From this point on, the client is no longer involved in any computation.

6. UFace servers

UFace system utilizes two servers: the data server and the key server, which are located in two different clouds to avoid

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

Table 1 – Notations.

F_1	Registered feature vector
F_2	Feature vector submitted for authentication
R_1	Value used to randomize F_1
R_2	Value used to randomize F_2
F_{1R}	Randomized F_1
F_{2R}	Randomized F_2

potential collusion. The data server is used for storing the encrypted UFace Pass for each user, i.e., the encrypted feature vector. The key server is used for maintaining keys that can decrypt users' encrypted feature vectors. Table 1 summarizes the notations used in the discussion. In what follows, we present the protocols during the registration and authentication phases, respectively.

6.1. *Registration*

The registration phase is very fast without much computation. Fig. 8 illustrates the main communication among users and servers during the registration protocol. Note that all communications are through secure channels which are not shown in the figure. At the beginning of the registration, the user (i.e., the UFace app) sends a new user ID to the web service provider. Once the web service provider verifies the uniqueness of the user ID, it will inform the UFace key server to send its public key (*PK*) to the user for encrypting the UFace Pass, and inform the UFace data server to prepare an Index ID for a new user. Then, the UFace data server will send the Index ID back to the web service provider which will forward it to the user. Upon receiving the public key *PK* and the Index ID, the user will encrypt the feature vector using the key *PK* to generate a UFace Pass, and then send the Index ID, UFace Pass to the data server. The data server will store the received user information under the corresponding web service folder and inform both the user and the web service provider the completion of the registration. Here, the data server does not see the user's real user ID.

6.2. *Privacy-preserving authentication*

6.2.1. *Authentication protocol overview*

After registration, a user can log onto the web service by simply providing his user ID and taking a close-up selfie which offers similar user experience as regular website login. Again, all communication is conducted through secure channels. The authentication protocol is outlined in Fig. 9.

First, the user (i.e., UFace app) sends the user ID to the web service provider. The web service provider will locate the Index ID of this user and forward the Index ID to the UFace data server to establish an authentication request. Then, the user will send his index ID and UFace Pass to the data server. Note that, since

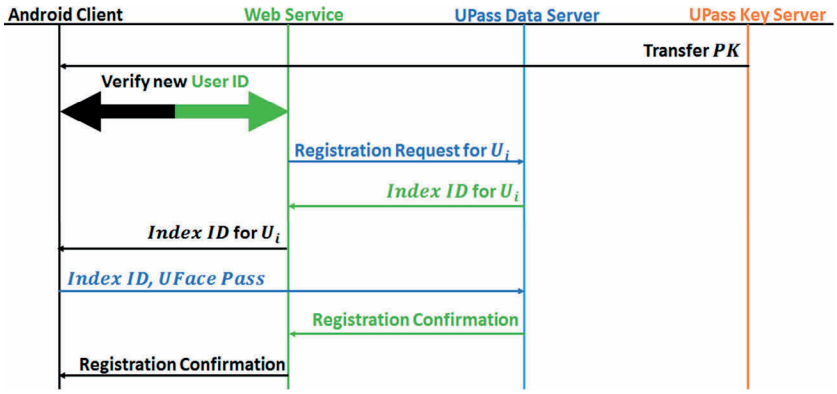


Fig. 8 – Registration Protocol.

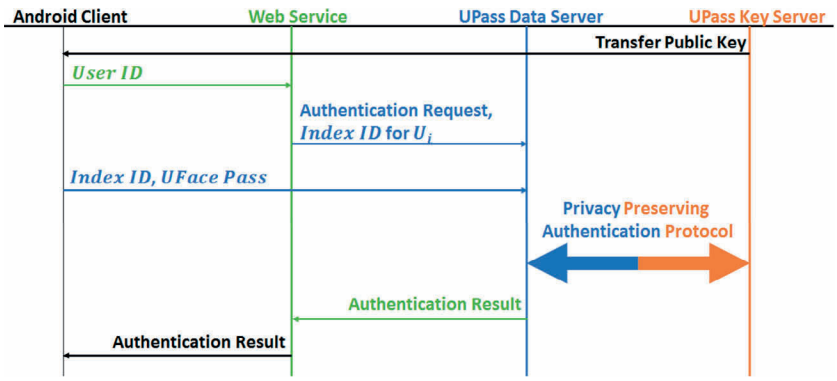


Fig. 9 – Authentication Protocol.

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

the UFace Pass is encrypted using the key server's public key, the data server will not be able to decrypt the UFace Pass. Upon receiving the user's authentication information, the data server will initiate a privacy-preserving authentication protocol with the UFace key server to jointly compare the registered user face feature vector and the received UFace Pass. Our proposed privacy preserving authentication protocol is built upon Garbled Circuit, and ensures that neither the data server nor the key server will see the plaintext of the user's feature vector. The details of the privacy-preserving authentication protocol is presented in the next subsection.

At the end of the privacy-preserving authentication protocol, the data server will return either "matched" or "unmatched" to the web service provider. "Matched" means the user's UFace Pass matches the registered information and hence the user is the legitimate user. Based on the data server's final message, the web service provider will grant the access to the user accordingly.

6.2.2. *Garbled circuit design for the face feature vector comparison*

We now proceed to describe the privacy-preserving authentication protocol between the data server and the key server. Our protocol leverages the garbled circuit techniques (Huang et al., 2011) because garbled circuit has been proven to be efficient for small functionality represented by a Boolean circuit and efficiency is a key requirement to achieve real-time authentication. In the following discussion, we will use F_1 to denote the feature vector that has been stored with the data server at the registration phase and F_2 to denote the feature vector received with the authentication request.

The design challenge is that garbled circuits can only handle plain text efficiently, but our feature vectors are all encrypted. In order to preserve efficiency, we need to feed decrypted data to the garbled circuits. If we send the encrypted feature vector directly to the key server for decryption, the key server will then know the user's photo information and hence violate the privacy preservation goal. To prevent this, our approach is to let the data server add random values R_1 and R_2 to feature vectors F_1 and F_2 using the Paillier encryption's additive property, and then send the randomized feature vectors to the key server. Now the key server can decrypt the randomized feature vectors but would not know the original photo information. These decrypted randomized feature vectors are the main input to the garbled circuit. Based on the homomorphic additive property of Paillier encryption (Paillier, 1999), the comparison results of the pair of randomized feature vectors would be the same as the original pair. In other words, we will still be able to know whether F_1 matches F_2.

Specifically, the data server sends the following information to the key server: R_1, R_2, R_{bit} and Th, whereby R_{bit} is a single bit used to hide the circuits outcome from the key server, and Th is an adjustable threshold value for face recognition accuracy. Then, the key server feeds the decrypted randomized feature vectors F_{1R} and F_{2R} to the garbled circuits. It is worth noting that at a high level view there are only these five inputs total, but in practice, there are multiple. Since each input is limited to the same bit size as the encryption key, multiple inputs are needed to represent each feature vector. For ease of understanding, each feature vector will be considered as one input in our discussion.

Algorithm 1 GCParser Circuit Code

Require: Data_Server: R_1, R_2, R_{bit}, and Th; Key_Server: F_{1R} and F_{2R}
Subtract R_1 from F_{1R}
Subtract R_2 from F_{2R}
Now F_1 and F_2 are in the circuit
Each bin b_{1i} of F_1 is isolated
Each bin b_{2i} of F_2 is isolated
for $i \leftarrow 1$ to $n * 59$ **do**
  $min_x = \min(b_{1i}, b_{2i})$
end for
$intersection = \sum_{x=1}^{n*59} min_x$
$pass = intersection \geq Th$
$result = pass \ \oplus R_{bit}$

The main steps of using garbled circuit to compare two encrypted feature vectors are outlined in Algorithm 1. At steps 1 and 2, the random values are subtracted from the randomized feature vectors. To speed up the process, the random values are inverted when provided to the garbled circuit, instead of being subtracted. The result will overflow the value to have the effect of modular division since the overflow bit is lost. This functions identically to subtraction, but faster. For clarity however, the random values are stated as being subtracted.

As a result of the first two steps, the circuit will have the original non-randomized feature vectors F_1 and F_2. Conceptually each feature vector is a matrix. Internally for the garbled circuit, each feature vector is the individual bins b_{ji} (where $j \in \{1, 2\}$ and $i \in \{1, 2, \ldots, 59 \times n\}$) from each $\bar{v}$ concatenated end to end. Thus, each feature vector has the internal appearance of $b_{j1}b_{j2}b_{j3}\ldots b_{59\times n}$ where n is the total number of $\bar{v}$. To further process these individual bins, the bins have to be separated, which is the main purpose of this step. Since each bin has a known bit size, the bins can be separated by linearly traversing the feature vector and isolating every block of the bit size. Once each bin is isolated, the intersection calculations begin. As shown in step 3 of the algorithm, by linearly walking through all the bins, the minimum between the corresponding bins of each feature vector is calculated.

In the next step, the sum of the minimums from each $\bar{v}$ are calculated. Inside the garbled circuit, this can be done in parallel to improve the efficiency. Then these partial sums are added together to find the final sum. Based on the final sum of the two feature vectors F_1 and F_2, we can now determine whether the feature vector submitted for authentication is similar enough to the one stored at the authentication servers to be considered as a match. For this, a similarity threshold needs to be defined according to the adopted face recognition algorithm. In our case, the threshold value is set to 0.9 based on the LBP algorithm, whereby 0.9 means that if two feature vectors have more than 90% of match, they are considered "match".

Finally, to prevent the key server from knowing the authentication result, the result is XOR'ed with R_{bit}. What the key server will see is a single bit that has a 50% chance of indicating "match" or "unmatch". On the other hand, the data server can

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

perform the XOR operation on the result and receive the final decision.

An overview of the protocol is given in Algorithm 2. When the protocol begins execution on the server side, it is assumed that both servers have a copy of the garbled circuit. The operations of the circuit do not change with each execution, so the circuit only needs to be constructed at the initial server setup. When GCParser is run the circuit file, the circuit will be garbled, so with each execution of the protocol, a different garbled circuit unique to that run is produced. Should either server attempts to change the circuit file, GCParser will abort operations due to these differences.

7. Security analysis

Our UFace system does not leak any user's biometric information to the data server, the key server or the web service provider. This is because our approach follows the security definitions in the literature of Secure Multi-party Computation (SMC) (Ben-David et al., 2008; Canetti, 2000; Goldreich, 2004; Goldreich et al., 1987; Katz and Lindell, 2007; Yao, 1982, 1986). As a result, our proposed protocol can be proved to be secure under the semi-honest model of SMC by using the simulation argument (Goldreich, 2004) as follows.

Algorithm 2 Overall Protocol Between Authentication Servers

Require: Data_Server: $[F_1]$, $[F_2]$ and Th

1: Data_Server:
   (a) Randomly generate R_1 and R_2, and encrypt them to produce $[R_1]$ and $[R_2]$
   (b) Calculate $[F_{1R}] = [F_1 + R_1] = [F_1] * [R_1]$ and $[F_{2R}] = [F_2 + R_2] = [F_2] * [R_2]$
   (c) Generate a random bit R_{bit} and produce a garbled circuit input file using R_1, R_2, Th, and R_{bit}
   (d) Send $[F_{1R}]$ and $[F_{2R}]$ to Key_Server
2: Key_Server:
   (a) Decrypt $[F_{1R}]$ and $[F_{1R}]$ and write the values to a garbled circuit input file
   (b) Start GCParser as server using its input file and the circuit
3: Data_Server:
   (a) Use GCParser to connect to the circuit running on Key_Server as a client using its input file
4: Data_Server and Key_Server:
   (a) Using GCParser, collaboratively evaluate the garbled circuit, and the evaluation result returns to both parties
5: Data_Server:
   (a) Perform XOR operation with the result and R_{bit}
   (b) Inform the authentication result to the web service: If the XOR result is a 1, authentication passed, else it failed

Definition 2. Let T_i be the input of party i, $\prod_i(\pi)$ be i's execution image of the protocol π and s be the result computed from π. π is secure if $\prod_i(\pi)$ can be simulated from $\langle T_i, s\rangle$ and distribution of the simulated image is computationally indistinguishable from $\prod_i(\pi)$.

In our case, the execution image for the data server mainly includes the two encrypted feature vectors. As mentioned earlier, since the data server does not have the private/decryption key and the encryption scheme is semantically secure, the image is computationally indistinguishable from a random sequence. Therefore, no information regarding the user's private data is leaked to the data server before executing the garbled circuit. Similar argument applies to the key server because the information that it received is randomized. In addition, all the intermediate results are either encrypted or randomized, and the garbled circuit approach is secure under the semi-honest model. As a result, based on the composition theorem (Goldreich, 2004), the overall protocol (Algorithm 2) is secure under the semi-honest model, i.e., any information regarding any users' private data is never leaked during the execution of our proposed protocol.

In the above definition, an execution image generally includes the input, the output and the messages communicated during an execution of the protocol. To prove the protocol is secure, we just need to show that the execution image of a protocol does not leak any information regarding the private inputs of participating parties (Goldreich, 2004). In our case, the execution image for the data server mainly includes the two encrypted feature vectors. As mentioned earlier, since the data server does not have the private/decryption key and the encryption scheme is semantically secure, the image is computationally indistinguishable from a random sequence. Therefore, no information regarding the user's private data is leaked to the data server before executing the garbled circuit. Similar argument applies to the key server because the information that it received is randomized. In addition, all the intermediate results are either encrypted or randomized, and the garbled circuit approach is secure under the semi-honest model. As a result, based on the composition theorem (Goldreich, 2004), the overall protocol (Algorithm 2) is secure under the semi-honest model, i.e., any information regarding any users' private data is never leaked during the execution of our proposed protocol.

Next, we discuss how our UFace system will react to common types of attacks including impersonation attack, man-in-the-middle attack, malleability attack, replay attack and denial-of-service attack.

7.1. *Impersonation attack*

Impersonation atttack is the most concerning attack in face authentication whereby the attacker tries to use the user's photo to gain access to the user's web accounts (Duc and Minh, 2009). To perform such attacks on our UFace system, the attacker needs to have the targeted user's *UserID* and the user's selfie. The user and the web service are the only 2 parties that know the user's *UserID*. We assume that the web service provider is responsible for its own security since if the attacker compromises the web service provider, the attacker directly gains all control of the user's account and no authentication is needed. Even so, it is worth noting that the attacker still would not have the users' selfies to masquerade as the user in other web

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

services. We also assume that the user's phone has an up-to-date operating system and anti-virus software. Moreover, to further prevent the attacker from collecting authentication information on the user's phone, all the user-end authentication can be performed in the secure zone on the phone. Also, the Android app deletes the photo used to generate the *UFacePass* after each authentication attempt.

We now discuss the scenarios when the attacker breaks into the data server or the key server since these two servers are located in a cloud and may be less protected. The data server possesses only an *IndexID* corresponding to the user's *UFacePass* and the key server has nothing with respect to the *UserID*. By compromising these two authentication servers, the attacker still would not be able to guess the user's *UserID* from the *IndexID* since the *IndexID* is basically a memory address in the data server.

Considering a more advanced attack whereby the attacker obtains the *UserID* via other means such as looking over the user's shoulders during use, our UFace system still prevents the attacker from obtaining the user's selfies for authentication. First, no party stores photos of the user or unencrypted feature vectors of the photos. Second, if the attacker tries to crop the user's face photo from photos published in social websites or takes the user's photo in a distance without being noticed by the user, these photos would not match the user's close-up image. The reason for this is that the close-up photos taken for authentication not only carry properties of the user's camera, but also use a focal point closer to the user. This means close-up images are seen as longer while zoomed-in images are seen as rounded (see Fig. 1). Our experiments with 20 users have proved that the LBP algorithm is capable of distinguishing these 2 types of images.

7.2. Man-in-the-middle attack

This is an attack where the attacker acts in-between the user and the authentication servers trying to fool each party into thinking they are directly communicating with each other. Many existing techniques, such as Public Key Infrastructures, can be adopted to help users verify the genuine authentication servers when establishing the secure communication channel. For example, the user encrypts the session key using the server's public key. Then, only the genuine server would be able to decrypt it and obtain the session key which will be used for the subsequent communication between the user and the server. Thus, the man-in-the-middle attack can be prevented.

7.3. Malleability attack

An encryption algorithm is malleable if it is possible for an adversary to transform a ciphertext into another ciphertext. Our protocol is robust against this because we adopt the secure communication channel established using AES encryption which has been proven to not be malleable.

7.4. Replay attack

If an attacker is able to obtain the user's Index ID and encrypted UFace pass, the attacker can launch the replay attack to gain access to the user's account. By communicating through the secure channel, outsiders cannot intercept the user's Index ID or UFace pass. The only person other than the user who can see the index ID and UFace pass is the data server. If the data server is compromised by an attacker, the attacker still needs to know the user ID from other means (such as social engineering) to be able to launch the replay attack. Even so, since the attacker does not obtain the plaintext of the user's biometric data, such replay attack once detected can be easily blocked by changing the key server's keys and re-register users. Our approach incurs much less damage compared to existing approaches whereby the attacker gains the plain-texts of users' biometric data and hence the users would not be able to re-use their biometric data as passwords any more.

7.5. Denial-of-service attack

Denial-of-Service (DoS) attack may occur when one of the authentication servers is compromised. Specifically, if only the data server is compromised by an attacker, the attacker can purposely fail all authentication requests by sending over incorrect feature vector pairs to the key server. Similar to the replay attack, DoS attack once detected can be remedied by reconfiguring the system. If only the key server is compromised by an attacker, the attacker can temper the authentication result such as flipping the final bit of the authentication decision which may result in incorrect authentication. Such attack does not cause permeant damage either once it is detected. If both servers are compromised by the same attacker, the attacker would gain the plaintexts of the users' feature vectors but still does not have the full knowledge of the users' face photos. Once such attack is detected, the whole system can be reconfigured by changing keys and feature vector sizes to function normally again.

8. Experimental study

In this section, we first introduce our experimental settings and evaluation metrics, and then report our experimental results.

8.1. Experimental settings

Our UFace system consists of an Android app for the client side and the security protocols at the server side. The UFace app was tested on an Android One Plus Three device. This device has a 16 MP rear camera and 8 MP front camera, uses the Snapdragon 820 processor (4 cores: 2×2.15 GHz and 2×1.6 GHz), and contains 6 GB RAM. The web service provider, the key server and the data server were simulated as three virtual machines that are running OpenSUSE 13.2 with access to 2 processor cores and 2 GB of RAM each. The virtual machines are hosted by a server with Intel Xeon processor (6 cores at 3.5 GHz) and 8.5 GB of RAM.

The performance of the UFace system is evaluated using two metrics: (i) face recognition accuracy, and (ii) authentication response time. Table 2 summarizes the parameters used, where the encryption key size is 1024 bits, the default grid size used for face recognition is 4×4, and the default photo size

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016

Table 2 – Experimental settings.

Parameters	Values
Number of users	**20**
Number of photos	**800**
Encryption key size	**1024 bits**
Grid size	2×2, **4×4**, 8×8
Photo Size	128×128, **256×256**, 512×512, 1024×1024

The default values are highlighted in bold.

Table 3 – Face recognition accuracy.

Authentication result	Own close-up photo	Own zoom-in photo	Others' close-up photo	Others' zoom-in photo
Pass	20/20	0	0	0
Fail	0	20/20	19/19	19/19

is 256×256. In what follows, we report the detailed experimental results.

8.2. *Accuracy analysis*

In the first round of experiments, we aim to examine if the UFace system is able to distinguish the user's close-up image (used for authentication) from the user's zoom-in image and other users' photos. For this, we collected close-up photos and zoom-in photos from 20 people. Specifically, each person was first asked to take two close-up selfies (such as the one shown in Fig. 1(a), one for registration and one for authentication. Then, we took another two photos for each person in a distance (such as the one shown in Fig. 1(b) using the zoom-in function of the phone, and these photos are called "zoom-in" photos. As a result, we collected a total of 800 photos from different brands of phones.

Then, for each user u_i, we use u_i's own close-up photo, u_i's zoom-in photo, and the other 19 users' close-up and zoom-in photos to try to log onto u_i's account. Table 3 summarizes the authentication results of the 20 users. Here, "Pass" refers to that the photo passed the authentication, i.e., the matching score is above the threshold which is 0.9 based on LBP; "Fail" refers to authentication failure, i.e., the access to the user account would be denied. As shown in the table, the UFace system is able to authenticate all the 20 users' close-up photos (denoted as 20/20), and denies the same user's zoom-in photo and other users' close-up and zoom-in photos (denoted as 19/19 as there are 19 other users). This means the UFace system is able to authenticate only the user's close-up photo but not any other photos including the same user's zoom-in photo and other users' photos. The reasons are two-fold. First, the UFace system adopts the LBP algorithm which has been proven to have very high recognition accuracy (over 95%) already. Second, the close-up photo does have the difference from the zoom-in photos caused by the optical distortion and pixel insertion in the zoom-in function. Therefore, the LBP algorithm can distinguish them.

8.3. *Effect of the total number of users*

In this set of experiments, we measure the response time of a single authentication request by varying the total number of registered users from 10 to 10,000. Fig. 10 shows the total response time which is the sum of client-side and server-side execution time. As we can see in Fig. 10, the total response time starting from a user taking a photo to receive the authentication decision is less than 5 seconds, which is comparable to common approaches that require users to type in usernames and passwords. This is probably the fastest privacy-preserving face authentication in the literature.

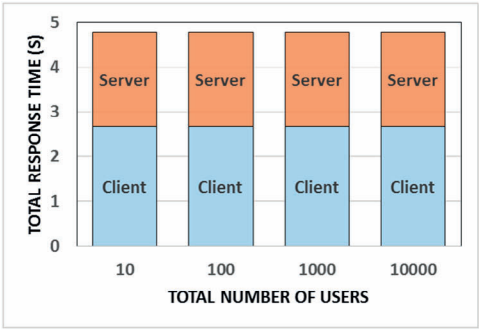


Fig. 10 – Effect of the total number of users.

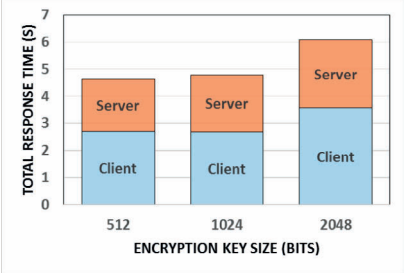
Moreover, the total response time remains constant as the number of users stored on the system increases. It is straightforward that the client side execution is not affected by the number of users stored at the server. At the server side, the constant performance is achieved attributed to the fact that a user's encrypted feature vector created at registration is stored along with the user's pseudo ID in a hash table. Thus, when a user attempts to authenticate, the user's pseudo ID provides the data server with a direct access to the user's entry in the hash table. As long as the hash table resides in the server's main memory, the authentication time would be constant regardless the total number of users. It is worth noting that when multiple users send authentication requests at the same time, each request will be handled by a separate thread. As long as the load does not exceed the server's computing capabilities, the response time will also be constant. The experimental results indicate that our system can be easily scaled up to thousands of millions of users by simply adding more data servers to handle different portions of the hash table.

8.4. *Effect of the encryption key size*

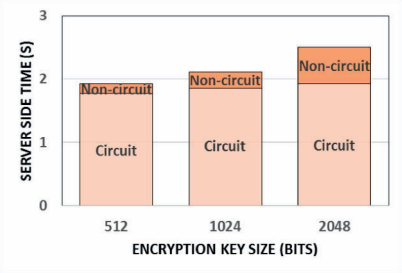
In the second round of experiments, we evaluate the effect of the different encryption key sizes. The commonly adopted key size is 1024 bits, and we also test 512 bits and 2048 bits. In the experiments, each user has 10 encrypted facial images stored at the data server for the face comparison. Each image is of 256×256 pixels and divided into a 4×4 grid prepared by the Android application. For each key size, we repeat 100 runs of tests and record the average response time. The results are shown in Fig. 11(a). As expected, the response time increases when the key size becomes bigger. This is because both the client-side application and the server-side security protocols need to compute larger encryption and hence takes more time.

The server-side execution time can be further divided into two categories: circuit processing time (denoted as "circuit" in the figure) and others (denoted as "non-circuit"). The circuit

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016



(a) Total response time



(b) Server-side processing time

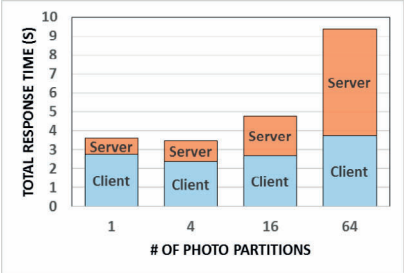
Fig. 11 – Effect of the encryption key size.

processing time refers to the time needed exclusively by the garbled circuit to execute. As we can see from Fig. 11(b), the increase in key size does not significantly impact the circuit execution time. When going from a key size of 512-bits to 2048-bits (i.e., 4 times of increase on size) only resulted in a 8.7% increase in average runtime. The reason is that even though the key size increases a lot, the amount of data processed by the garbled circuit does not change much. Given the same image and same grid partition, the number of bins generated by the local binary pattern algorithm is the same. The only difference is that each bin is stored in either 512, 1024, or 2048 bit numbers depending on the key size. Unlike the constant circuit time, the non-circuit time increases more with the increase of the key size, which is likely due to the encryptions and decryptions performed along with homomorphic addition. At 2048-bit key sizes, when the encrypted feature vectors are randomized, the multiplication between two 2048 bit numbers is performed. This creates a significant overhead when compared to the multiplication of 512-bit numbers.

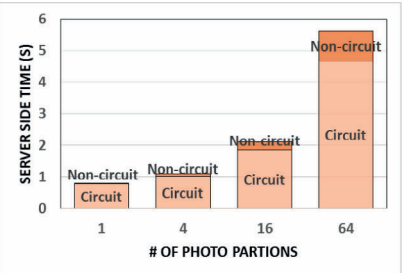
8.5. *Effect of grid size*

We now proceed to evaluate the impact of the grid size on the response time. According to the face detection algorithm (Ahonen et al., 2004), too few or too more grids could both affect the face recognition accuracy and efficiency. Given a photo of 256×256 pixels, we tested the following partitions: 1 partition, 4 partitions (2×2), 16 partitions (4×4) and 64 partitions (8×8). The reason for choosing these grid sizes is that one requirement of local binary patterns is to divide an image into an n-by-n grid. Since the images being processed are 256×256 pixels, the n values of 3, 5, 6 and 7 do not evenly divided 256. Thus, to avoid fractional pixels, the values of 1, 2, 4 and 8 were chosen. Fig. 12(a) reports the total response time. As we can see that, the client-side processing time first decreases and then increases when the number of partitions increases. This is mainly due to the design of the original face recognition algorithm, whereby the grid size is directly tied to the total number of bins and the size of bins inside the feature vector. Given that each partition has the same number of bins, the more partitions, the more total number of bins and hence more computation. However, the more partitions, the smaller the bit-size of the bins which helps reduces the computation. For example, given a single partition, there are 59 bins; given 16 partitions, there are 3776, whereby the total number of bins is $59 \times grid\_size$. When there is only one partition, in the worst case, all the pixels through local binary patterns could fall into one bin, such that the bin needs to be 17-bits to store the value of 65536. On the other hand with 64 bins, each grid at worst case will have 1024 pixels falling into a single bin and requires 11-bit bins. Combining these two effects, we see the concave up curve for the total response time.

Fig. 12(b) provides a deeper look at the server side, from which we can see that the execution of the garbled circuite dominates the processing time and keeps increasing with the number of partitions. This is probably because that the number of bins has bigger impact versus grid size. Specifically, a single partition has 59 bins which requires 59 minimum comparisons



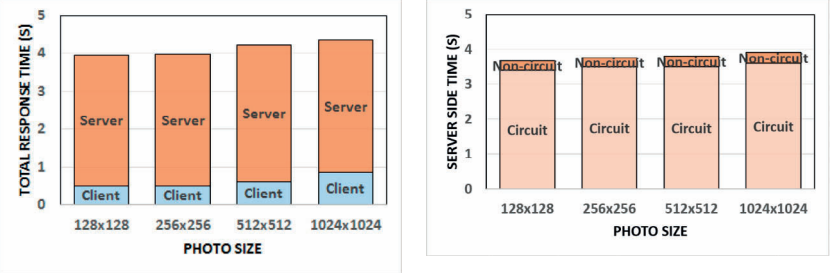
(a) Total response time



(b) Server-side processing time

Fig. 12 – Effect of the number of partitions.

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016



(a) Total response time (b) Server-side processing time

Fig. 13 – Effect of photo size.

and 59 values summed. In the case of 16 partitions, there are 3776 bins which equate to 3776 minimum comparisons and 3776 values summed. With this direct correlation between the number of bins and the number of operations internally in the garbled circuit, it is not surprising to see that the time increases in a nearly linear fashion.

8.6. *Effect of photo size*

In the last round of experiments, we evaluate the effect of the photo size. The original photos obtained from various phones are resized for testing. Fig. 13(a) shows the total response time when the resized photo size varies from 128×128 to 1024×1024. As expected, the more pixels in the photo, the more computations are needed, leading to increase of the processing time at both the client and the server-side. However, the increase at the server side is not that significant as shown in Fig. 13(b). The reason is similar to that discussed in the previous experiments. The performance of the garbled circuit is mainly dominated by the number of partitions and the total number of bins.

9. Conclusion

In this paper, we present a privacy-preserving face authentication system, called UFace, for authenticating web services. UFace is unique in that it helps users authenticate with web service providers without disclosing the actual content of their facial images to any party including web servers and authentication servers. Along with the use of special close-up images, UFace successfully prevents the common threat from the impersonation attack using online images. The UFace system has been implemented at Android phones for performance evaluation. The experimental results has demonstrated that the UFace system is capable of fulfilling the authentication task accurately and efficiently within seconds.

Acknowledgment

The work is funded by National Science Foundation under project DGE-1433659, CNS-1651455 and CNS-1564101.

REFERENCES

Ahonen T, Hadid A, Pietikäinen M. FACE recognition with local binary patterns. In: Computer vision-eccv 2004. Springer; 2004. p. 469–81.

Belhumeur PN, Hespanha JP, Kriegman DJ. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. IEEE Trans Pattern Anal Machine Intell 1997;19(7):711–20.

Ben-David A, Nisan N, Pinkas B. Fairplaymp – a system for secure multi-party computation. In Proceedings of the ACM computer and communications security conference (ACM CCS), October 2008.

Blanton M, Aliasgari M. Secure outsourced computation of iris matchin. J Comput Security 2012;20(2-3):259–305.

Blanton M, Gasti P. Secure and efficient protocols for iris and fingerprint identification. In: Computer security–ESORICS 2011. Springer; 2011. p. 190–209.

Bringer J, Chabanne H, Patey A. Privacy-preserving biometric identification using secure multiparty computation: an overview and recent trends. IEEE Signal Process Mag 2013;30(2):42–52.

Canetti R. Security and composition of multiparty cryptographic protocols. J Cryptol 2000;13(1):143–202.

Chun H, Elmehdwi Y, Li F, Bhattacharya P, Jiang W. Outsourceable two-party privacy-preserving biometric authentication. In Proceedings of the 9th ACM symposium on Information, computer and communications security, pages 401–412. ACM; 2014.

Duc NM, Minh BQ. Your face is not your password. In Black Hat Conference, volume 1; 2009.

Erkin Z, Franz M, Guajardo J, Katzenbeisser S, Lagendijk I, Toft T. Privacy-preserving face recognition. In: Privacy enhancing technologies. Springer; 2009. p. 235–53.

Evans D, Huang Y, Katz J, Malka L. Efficient privacy-preserving biometric identification. In Proceedings of the 17th conference network and distributed system security symposium, NDSS; 2011.

Goldreich O. Chapter encryption schemes. In: The foundations of cryptography, vol. 2. Cambridge University Press; 2004.

Goldreich O, Micali S, Wigderson A. How to play any mental game – a completeness theorem for protocols with honest majority. In 19th ACM symposium on the theory of computing, pages 218–229, New York, New York, United States; 1987.

Goldwasser S, Micali S. Probabilistic encryption. J Comp Syst Sci 1984;28(2):270–99.

Huang Y, Evans D, Katz J, Malka L. Faster secure two-party computation using garbled circuits. In the 20th USENIX security symposium, August 2011.

Internet Live Stats. Total number of Websites; 2016. Available from: http://www.internetlivestats.com/total-number-of-websites/.

Katz J, Lindell Y. Introduction to modern cryptography. CRC Press; 2007.

Katz J, Sahai A, Waters B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Advances in cryptology–EUROCRYPT 2008. Springer; 2008. p. 146–62.

Li Y, Li Y, Yan Q, Kong H, Deng RH. Seeing your face is not enough: an inertial sensor-based liveness detection for face authentication. In: Proceedings of the 22Nd ACM SIGSAC conference on computer and communications security. New York, NY, USA: ACM; 2015. p. 1558–69 CCS '15.

Li Z, He W, Akhawe D, Song D. The emperor's new password manager: security analysis of web-based password managers. In 23rd USENIX security symposium (USENIX security 14), pages 465–479; 2014.

Luo P. Hierarchical face parsing via deep learning. In Proceedings of the 2012 IEEE conference on Computer Vision and Pattern Recognition (CVPR); 2012.

Osadchy M, Pinkas B, Jarrous A, Moskovich B. Scifi-a system for secure face identification. In Security and Privacy (SP), 2010 IEEE Symposium on, pages 239–254. IEEE; 2010.

Paillier P. Public key cryptosystems based on composite degree residuosity classes. In Advances in Cryptology – Eurocrypt '99 Proceedings, LNCS 1592, pages 223–238, Prague, Czech Republic, May 2–6. 1999.

Pal D, Khethavath P, Thomas JP, Chen T. Secure and privacy preserving biometric authentication using watermarking technique. In: Security in computing and communications. Springer; 2015. p. 146–56.

Sadeghi A-R, Schneider T, Wehrenberg I. Efficient privacy-preserving face recognition. In: Information, security and cryptology–ICISC 2009. Springer; 2010. p. 229–44.

Sedenka J, Govindarajan S, Gasti P, Balagani KS. Secure outsourced biometric authentication with performance evaluation on smartphones. IEEE Trans Info Forensics Sec 2015;10(2):384–96.

Shamir A. How to share a secret. Commun ACM 1979;22(11):612–13.

Shen E, Shi E, Waters B. Predicate privacy in encryption systems. In: Theory of cryptography. Springer; 2009. p. 457–73.

Sun Y, Wang X, Tang X. Deep convolutional network cascade for facial point detection. In 2013 IEEE conference on computer vision and pattern recognition, pages 3476–3483; 2013.

Tagat A. Online fraud: too many accounts, too few passwords. 2012.

Tan X, Triggs B. Enhanced local texture feature sets for face recognition under difficult lighting conditions. IEEE Trans Image Process 2010;19(6):1635–1650.

Turk M, Pentland A. Eigenfaces for recognition. J Cogn Neurosci 1991;3(1):71–86.

Yao AC. Protocols for secure computation. In Proceedings of the 23rd IEEE symposium on foundations of computer science, pages 160–164. IEEE; 1982.

Yao AC. How to generate and exchange secrets. In Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, pages 162–167. IEEE; 1986.

Dan Lin is an associate professor at the Department of Computer Science of Missouri University of Science and Technology. She is also the Director of Cybersecurity Lab at Missouri S&T. She received her BS degree in Computer Science from Fudan University, Shanghai, China, in 2002, and she received her PhD degree in Computer Science from the National University of Singapore in 2007. She was a post doctoral research associate at Purdue University for two years. Her main research interests cover many areas in the fields of database systems, information security, cloud computing, and vehicular ad-hoc networks.

Nicholas Hilbert received his BS degree in Computer Engineering in 2015 and his MS degree in Computer Science in 2017 at Missouri University of Science and Technology. His research interest is secure multi-party computation.

Christian Storer is a PhD student at the Department of Computer Science at Missouri University of Science and Technology. He received his BS degree in Computer Science in 2015. His research interest is privacy preserving data mining.

Wei Jiang is an associate professor at the Department of Computer Science of Missouri University of Science and Technology. He is also the associate chair of graduate study in the CS department. He received his BS degree in Computer Science from Iowa University in 2002 and received his PhD degree in Computer Science from Purdue University in 2008. His main research interest is applied cryptography.

Jianping Fan is a professor at UNC-Charlotte. He received his MS degree from Northwestern University, China in 1994 and his PhD degree from Shanghai Institute of Optics and Fine Mechanics, Chinese Academy of Sciences in 1997. He was a postdoc at Fudan University, China, during 1997-1998. From 1998 to 1999, he was a Researcher with Japan Society of Promotion of Science, Department of Information System Engineering, Osaka University. From 1999 to 2001, he was a postdoc in the Department of Computer Science at Purdue University. His research interests include image privacy protection, automatic image/video understanding, and statistical machine learning.

Please cite this article in press as: Dan Lin, Nicholas Hilbert, Christian Storer, Wei Jiang, Jianping Fan, UFace: Your universal password that no one can see, computers & security (2017), doi: 10.1016/j.cose.2017.09.016