UFace: Your Universal Password That No One Can See

Nicholas Hilbert
Missouri University of Science and Technology
Rolla, MO, USA
Email: nsh9b3@mst.edu

Christian Storer
Missouri University of Science and Technology
Rolla, MO, USA
Email: cs9yb@mst.edu

Dan Lin
Missouri University of Science and Technology
Rolla, MO, USA
Email: lindan@mst.edu

Wei Jiang
Missouri University of Science and Technology
Rolla, MO, USA
Email: wjiang@mst.edu

***Abstract*—With the advantage of not having to memorize long passwords, people are more interested in adopting face authentication for use with mobile devices. However, since facial images are widely shared in social networking sites, it becomes a challenging task to securely employ face authentication for web services to prevent attackers from impersonating the legal users by using the users' online face photos. Moreover, existing face authentication protocols either require users to disclose their unencrypted facial images to the authentication server or require users to execute computationally expensive secure multi-party computation protocols. For mobile devices with limited computational power, this presents a problem that cannot be overlooked. In this paper, we present a novel privacy preserving face authentication system, called UFace, which has users take close-up facial images for authentication to prevent against impersonation attacks of users' online facial images. UFace also guarantees that the facial images are only seen by the users and not by any other party (e.g., web service providers and authentication servers). UFace was implemented through two facets: an Android client application to obtain and encrypt the feature vector of the user's facial image, and server code to securely authenticate a feature vector across multiple servers. The experimental results demonstrate that UFace not only can correctly authenticate a user, but also can be done within seconds which is significantly faster than any existing privacy preserving authentication protocol.**

I. Introduction

With around 1 billion websites online today [1], statistics [24] show that each Internet user has an average of 26 different online accounts, with individuals between the age of 25 to 34 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 or changed ever so slightly due to the challenge of memorizing many different passwords. The surprising fact is that a person uses, on average, just 5 unique passwords for all their accounts [24]. 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.

The question this paper aims to solve: Is there a way that does not require individuals to memorize many different passwords while still preventing attackers from accessing confidential information? Face authentication is one potential solution to this. This tool means users will only need to send an image of their face (or a feature vector representing their face) to prove their identities - much easier than trying to remember the password that correlates with the service being used. Since face authentication is a relatively new technology, it still needs to overcome several critical challenges: maintaining high accuracy of authenticating a user's face, preventing masquerade attacks by using old images, and preserving privacy of users' information that have been used for authentication. The accuracy of authenticating based on facial recognition is no longer a major concern since certain algorithms can achieve an accuracy of over 90% [25]. However, the remaining two challenges have not been well addressed. Specifically, in existing face authentication systems, it is possible for attackers to reuse the photos obtained from social networks and then be authenticated as the photo owners. To prevent such impersonation, the latest technique is face liveness detection [16]. Unfortunately, the face liveness detection approach has recently also been proved to be vulnerable by researchers [27] who can create realistic 3D facial models with a handful of pictures from social media to spoof the face liveness detection.

To overcome these security and privacy challenges during face authentication and enable its wide adoption in web services, we propose a novel privacy-preserving face authentication system, called UFace, where "U" stands for both "your" and "universal". Our main idea is to let users take close-up images only for authentication purposes using their own mobile devices. 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. Then, the next step is to keep such close-up images safe with the users so that no one could

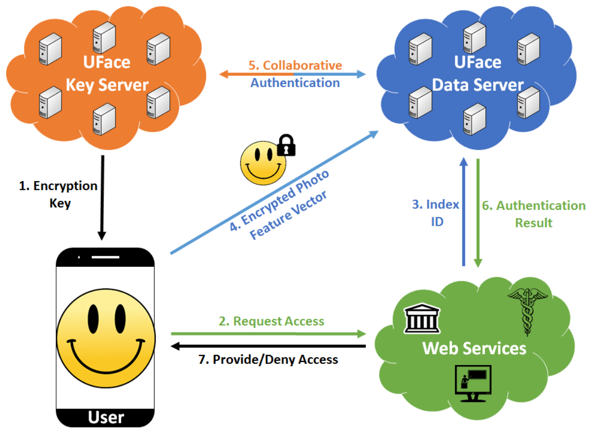


Fig. 1. UFace System - Authentication Overview

gain access or reuse them for authentication. To achieve this, we have designed an efficient, secure, and privacy-preserving authentication protocol that never discloses the plaintext of the close-up images (or their respective feature vectors) to the authentication servers and web service providers during the entire authentication process. It is worth noting that our work is unique compared to existing privacy-preserving face authentication approaches [10] which all require the authentication server to maintain plaintext facial images. If using the existing face authentication protocols to authenticate close-up facial images, an attacker will still be able to obtain the users' close-up images after compromising the authentication servers and impersonate the users later on. In our UFace system, even if the attacker compromises the authentication servers, he/she will only obtain encrypted facial image feature vectors and cannot reuse them for authentication (detailed security analysis will be presented in Section VIII).

As shown in Figure 1, the UFace system involves four parties: (1) *end users*, (2) *web service providers*, (3) *UFace data servers*, and (4) *UFace key servers*. UFace is a third party that hosts two authentication servers to facilitate privacy-preserving authentication between web service providers and end users through mobile devices. An UFace application will be installed on the user's Android device. When the user wants to log into a web service (already registered with UFace), the user just needs to take a photo while the app take cares of everything else. The UFace system will then carry out a secure multi-party computation to authenticate the user with the web service. Our technical contributions are summarized as follows:

- UFace is built in a multi-cloud environment and can serve for multiple web services simultaneously. Its authentication protocol prevents the disclosure of any users' plaintext images or feature vectors to any party participating in the protocol: (1) *web services*, (2) *UFace data servers*, and (3) *UFace key servers*.
- UFace has an efficient homomorphic encryption based authentication protocol which allows the two UFace servers to collaboratively conduct facial image comparison on encrypted facial images. Due to the complexity of the facial matching algorithms, designing the collaborative homomorphic computation was challenging. It required mapping and integrating various types of encrypted computations to work alongside garble circuit operations. The overall process needed to be highly efficient so the response time to users would be comparable to logging in using common password strings.
- UFace has an Android application that is capable of efficient photo feature extraction and encryption and keeps each user's computational burden to minimum. The development of the Android application involved multiple challenges that dealt with the limited memory/computing power of these mobile devices along with the need to design a new library for facial feature generation tailored to work on mobile device.
- UFace has been evaluated both theoretically and experimentally. The security analysis shows that UFace is robust against various types of attacks. The experimental results demonstrate that UFace not only can correctly authenticate a user, but also can be done within seconds which is significantly faster than any existing privacy preserving authentication protocol.

The rest of the paper is organized as follows. Section II discusses the related works on privacy preserving face authentication and face recognition. Section III gives an overview into the tools UFace utilizes while Section IV provides an overview UFace's 2 phases. The threat model and security goals of UFace are analyzed in Section V. Then, Section VI presents the UFace Android application at the user side and Section VII presents the protocols at server side. Section VIII provides a security analysis of the system and Section IX reports the performance study. Finally, Section X concludes the paper.

II. Related Works

In this section, we first discuss related works on privacy-preserving biometric authentication and then briefly review the commonly used face recognition algorithms.

A. 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 [6]. One of the earliest attempts towards privacy preservation during biometric authentication is by Erkin et al. [10]. 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. [20] improved the efficiency of the above protocol. Following the similar settings, Osadchy et al. [17] 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. Huang et al. [11] proposed a secure protocol for fingerprint matching while Blanton et al. [5] proposed security protocols for both fingerprints and iris. Recently, Sedenka et al. [21] employed a similar idea and implemented the privacy-preserving face authentication on smartphones. 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 the each users 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 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 [15], [23] which is not as secure as the additive homomorphic encryption scheme adopted into UFace. Their multi-server protocol leverages a secret sharing scheme [22] and requires at least three independent servers, whereas our UFace system only needs two independent servers and is much more efficient. In [19], Pal et al. 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. [8] 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.

III. Authentication Tools

To accomplish authentication between the UFace servers, a few different types of tools are used: facial recognition and secure multi-party computations (SMCs). This section gives a brief overview of each of these UFace operations to better understand the implementation detailed in Section VII.

A. Face Recognition with Local Binary Patterns

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 [26] and Fisherfaces [3]. Later, a more advanced approach was proposed using so-called Local Binary Patterns (LBP) [2] to generate a feature vector for a photo. Face recognition algorithms using LBP patterns yield a higher accuracy rate under different environments (e.g., different lightings).

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 counter-clockwise pattern.

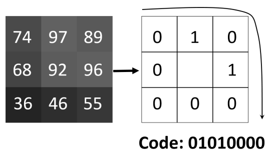


Fig. 2. An example of computing the LBP for a pixel

An example of the basic LBP operation is shown in Figure 2 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.

This current LBP scheme doesn't 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 $\{H_1 H_2 \ldots H_k\}$ 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.

B. 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's first encrypted using 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's 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's 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 [18].

C. 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 [28].

IV. System Overview

We designed UFace as a privacy preserving face authentication framework to prevent web service providers from gaining access to user's facial images or their respective feature vectors. To accomplish this, UFace serves as the middle man between multiple web service providers and users. As shown in Figure 1, there are 4 entities involved in the system: (1) *end users*, (2) *web service providers*, (3) *UFace data servers*, and (4) *UFace key servers*. The UFace data servers store users encrypted authentication information while the UFace key servers manage the key capable of decrypting this information. However, the 2 server clouds never collude about each other's information and execute a secure multi-party computation to authenticate users. This design follows the spirit of "separation of duty" to achieve privacy preservation. For the remainder of the paper, each UFace server cloud will be considered to be 1 single server for easy illustration of the main ideas. UFace is comprised of 2 main phases of operation: (1) *Registration* and (2) *Authentication*.

A. Registration

To register with a web service, a user just needs to install the UFace Android client. The user will select a web service that is registered with UFace and create a unique $UserID$ for the web service. To finish registration, the user only needs to take a close-up photo of their face. In the background, the app takes the taken photo, executes the LBP algorithm to generate a feature vector, and encrypts the feature vector before sending this information off to the UFace data server for authentication. The UFace data server receives the encrypted data and stores the information at a specific location - $IndexID$ - for that user (which is shared with the web service provider). Note, the UFace data server (or any other party) never has unencrypted images or feature vectors of the user. All operations are conducted on encrypted data only.

B. Authentication

To begin authentication for a web service, the user only needs to select the registered web service and take a close-up image of the user's face. While this is occuring, the app will send the the $UserID$ to the web service so it knows a specific user is attempting to login. The web service will then forward the associated $IndexID$ to the UFace data server so the data server knows what data to authenticate the user's information with. Meanwhile, the Android app has executed the LBP algorithm, generated a feature vector from the taken photo and encrypted this feature vector. Finally, once a message is received from the web service stating that authentication may begin, the app sends the encrypted feature vector and its $IndexID$ to the UFace data server. The UFace data and key servers collaboratively conduct a secure protocol to determine the comparison result between the sent encrypted feature vector and the one stored on the UFace data server at the location determined by the $IndexID$. The secure protocol ensures that each server's information remains confidential to each server, so even though the UFace key server has the key to decrypt all messages, it never obtains information about the user's biometric data. The result is then sent to the web service which then forwards the response to the user. Figure 1 provides an outline for how authentication works within the UFace system (numbers show order of information travel).

V. 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 protocol but may try to learn additional information by exploring the information available to them [12]. 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 [4], [7], [12], [14] are secure under the semi-honest model. In this model, the participating parties will not collude, and for our case, these parties are the UFace data and key servers. 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.

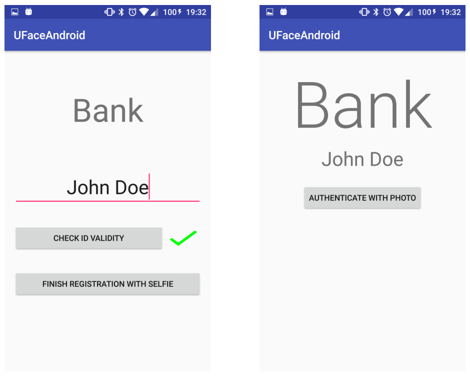
The security goal of our UFace system is to keep users' authentication information fully private, which includes the following aspects:

- Users do not need to reveal the actual content of their facial images to any party during the authentication process.
- Web service providers can safely outsource the authentication process to UFace without violating users' privacy concerns regarding their biometric data that have been used for authentication.
- UFace authentication servers perform authentication on fully encrypted data.
- 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.

VI. UFace Android Application

The UFace Android application consists of two modules: *Web Service Access* and *$UFacePass$ Generation*. The first refers to the 2 phases of UFace: registration and authenticaiton. The second module explains how the encrypted feature vector is generated. Each will be discussed in the following sections.



(a) Registration (b) Authentication

Fig. 3. Snapshots of UFace App

A. Web Service Access

To start logging into a web service securely, users only need to register with a web service provider. Upon startup, the app retrieves a public key (PK) used for encryption and displays a blank screen with a "+" icon. By clicking on the "+" icon, a new window will display 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, the user will see a common login page to create a unique "$UserID$" for the web service. The $UserID$ will be sent to the web service provider to verify the uniqueness (as shown in Figure 3(a)). If the $UserID$ is good to use, the web service provider will return a so-called "$IndexID$" to the user (which was obtained from the UFace data server). This $IndexID$ indicates the location where the user's "$UFacePass$" (encrypted feature vector) will be stored at the data server and also prevents the data server from knowing the user's actual $UserID$. By creating different $IndexID$s, it would be easy to extend current system to accommodate multiple user devices registered for the same web service. Finally, the user just needs to take a close-up facial photo to finish registration. The photo is used to generate the $UFacePass$ (the algorithm is presented in Section VI-B) and it's sent to the UFace data server along with the $IndexID$. The $UserID$, $IndexID$, and web service information are stored in the app if registration is successful.

The main page of the app will now show a list of icons representing registered web services. After selecting a web service, the user will be required to take a close-up facial photo (Figure 3(b)) to generate a new $UFacePass$. The app will send the $UserID$ to the web service provider while the $IndexID$ and the $UFacePass$ are sent to the UFace data server. If access is granted, the user will be directed to the account for that web service.

B. $UFacePass$ Generation

The most important feature of the UFace app is the $UFacePass$ generator. This converts the user's close-up photo into the encrypted feature vector efficiently. The overall process can be seen in Figure 4. However, there are 3 main steps to creating the $UFacePass$: (i) feature vector generation, (ii) feature vector manipulation and (iii) feature vector encryption.

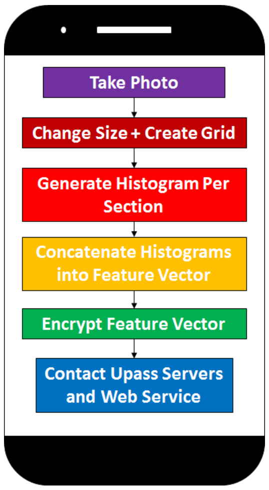

Take Photo
Change Size + Create Grid
Generate Histogram Per Section
Concatenate Histograms into Feature Vector
Encrypt Feature Vector
Contact Upass Servers and Web Service

Fig. 4. UPass Generation

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 smartphone as shown in Figure 8(a). Once the image is captured, the LBP algorithm is executed to create a feature vector. As introduced in Section III-A, a 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 $F_p = \langle \overrightarrow{v_1}, \ldots, \overrightarrow{v_k} \rangle$, where $\overrightarrow{v_i} = \langle b_1, \ldots, b_{59} \rangle$ $(1 \le i \le k,\ k = 16)$.

2) Feature Vector Manipulation: After generating the feature vector, the next step would be to encrypt it using a public key obtained from the key server. However, as stated in Section III-A, for an image (of size 256x256) broken into 16 sections, the number of bins that need to be encrypted is 944 and the max number of bits needed to represent each section is 13. The key size used for Paillier encryption in UFace is 1024 bits, which means that if each bin is encrypted separately, there would be 1024 - 13 = 1011 bits of wasted information with every encryption and the total size of the feature vector would be 944 $\times$ 1024 = 966,656 bits or 118 kB.

To improve this, bin values will be concatenated together into 1 value which is then encrypted. This would mean that $\lfloor \frac{1024}{13} \rfloor$ = 78 bins can be used in 1 encryption with 1024 mod 13 = 10 extra bits. Instead of 944 encryptions with 118 KB of data being sent, there is only $\lceil \frac{944}{78} \rceil$ = 13 encryptions with 13,312 bits or 1.625 KB. This is roughly a 72.6% speedup compared to encrypting every bin.

UFace does this efficiently be creating big integer values from the bits of 78 consecutive bins while padding the first

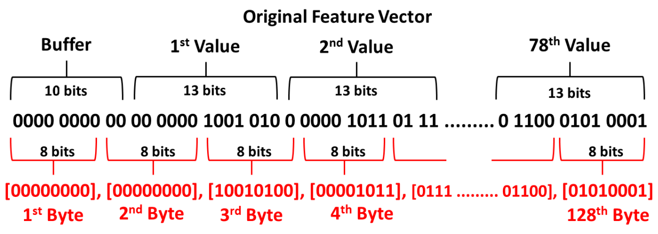


Fig. 5. Feature vector compaction

10 bits with 0s. The 13 big integers created are then encrypted to become the $UFacePass$.

3) Feature Vector Encryption: Now that an efficient feature vector is obtained, the final step is to actually encrypt the data. This is done using Paillier encryption (see Section III-B). This function simply iterates through the 13 big integer values obtained from the above data manipulation and encrypts each item. After the encryption of the feature vector is completed, the encrypted feature vector along with the $IndexID$ is sent to the UFace data server for registration or authentication. From this point on, the client is no longer involved in any computations.

VII. UFace Servers

UFace system utilizes a data server and a key server, which are located in two different clouds to avoid potential collusion. The data server is used for storing the encrypted $UFacePass$ for each user, i.e., the encrypted feature vector. The key server is used for maintaining the key (PK) that can decrypt users' $UFacePass$. In what follows, we present the registration and authentication phases, respectively.

A. Registration

The registration phase is very fast without much computation. Figure 6 illustrates the main communication between all parties during registration - note that all communications are through secure channels. Before registration begins, the UFace key server transfers PK to the user. Then the user (i.e., the UFace app) sends a new $UserID$ to the web service provider. Once the web service provider verifies the uniqueness of the $UserID$ it informs the UFace data server to prepare an $IndexID$ for a new user. The UFace data server will send the $IndexID$ back to the web service provider which will forward it to the user. Upon receiving the $IndexID$, the user will encrypt the feature vector using PK to generate a $UFacePass$ and then send the $IndexID$ and $UFacePass$ to the data server. The data server will store the received user information at the location provided from the $IndexID$ and inform the web service of a successful registration which then informs the user. The data server never sees the user's real $UserID$ and the $UFacePass$ is encrypted with the key stored on the UFace key server so it cannot decrypt the information.

B. Privacy-Preserving Authentication

After registration, an user can log onto the web service by simply selecting the web service on the Android app and

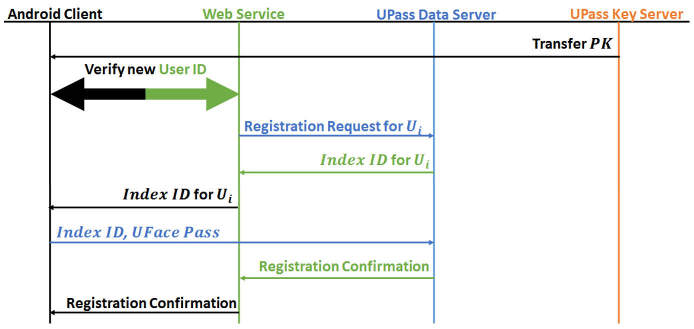


Fig. 6. Registration Protocol

taking a close-up selfie; this offers a similar user experience as common website login services. Again, all communication is conducted through secure channels. The authentication protocol is outlined in Figure 7.

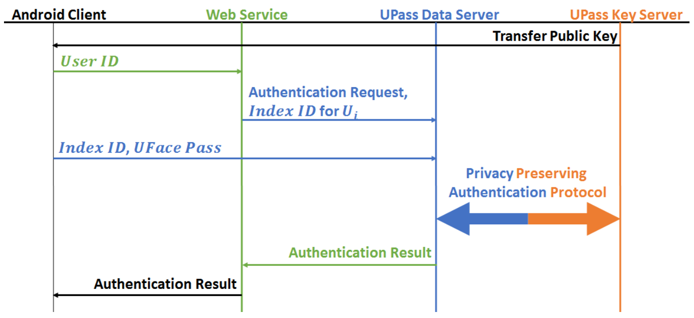


Fig. 7. Authentication Protocol

First, the user (i.e., UFace app) obtains PK from the UFace key server. Then the user sends the $UserID$ to the web service provider who will locate the $IndexID$ of this user and forward it to the UFace data server to establish an authentication request. Then, the user will send his $IndexID$ and the $UFacePass$ to the data server. 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 received $UFacePass$ with the user's registered $UFacePass$. Our proposed privacy preserving authentication protocol is built with garbled circuits, and ensures that neither the data server nor the key server will see the plaintext of the user's biometric information. The details of the privacy-preserving authentication protocol is presented in the next section.

At the end of the privacy-preserving authentication protocol, the data server will return the result to the web service provider. If the result is a match, then the user's $UFacePass$ matches the registered stored information over a specific threshold. Based on the result, the web service provider will grant/deny access to the user accordingly.

1) Garbled Circuit Design for 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 [13] because garbled circuits have 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 denote the two encrypted feature vectors as F_1 and F_2, whereby F_1 refers to the feature vector that has been stored with the data server at the registration phase and F_2 refers to 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 without learning about the user's information. These decrypted randomized feature vectors are the main input to the garbled circuit. Based on the homomorphic additive property of Paillier encryption [18], 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 garbled circuit: 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 $F_{2R}$
 1: Subtract $R_1$ from $F_{1R}$
 2: Subtract $R_2$ from $F_{2R}$
 3: Now $F_1$ and $F_2$ are in the circuit
 4: Each bin $b_{1i}$ of $F_1$ is isolated
 5: Each bin $b_{2i}$ of $F_2$ is isolated
 6: for $i \leftarrow 1$ to $k \times n$ do
 7:     $min_x = \min(b_{1i}, b_{2i})$
 8: end for
 9: $intersection = \sum_{x=1}^{k \times n} min_x$
10: $pass = intersection \geq Th$
11: $result = pass \oplus R_{bit}$
```

The main steps of using a 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. For the garbled circuit, each feature vector is the individual bins $b_{j,i}$ (where $j \in \{1,2\}$ and $i \in \{1,2,\ldots,n \times k\}$) from each $\overrightarrow{v}$ concatenated end to end. The value j represents 1 specific feature vector, k represents the number of sections an image is separated into while n is the number of bins for the histogram created in each section, and $\overrightarrow{v}$ is the histogram generated for each section. Thus, each feature vector has the internal appearance of $b_{j,1}b_{j,2}b_{j,3}\ldots b_{j,k\times n}$. 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 $\overrightarrow{v}$ are calculated which is done in parallel to improve efficiency. These minimums are added together to obtain the final sum. Based on the final sum of the two feature vectors F_1 and F_2, we can now determine whether the 2 feature vectors are similar enough to be considered a match. To determine similarity, the final sum needs to be compared against a threshold. In our case, the threshold value is set to 0.885 based on the results obtained in Section IX, which means the 2 feature vectors need to be 88.5% similar.

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". The data server can perform the XOR operation on the result to receive the actual result.

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 upon initial server setup. When GCParser runs the circuit file, the circuit will be uniqly garbled. Therefore, with each execution of the protocol, a different garbled circuit is produced. Should either server attempt to change the circuit file, GCParser will abort operations due to these differences.

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
```

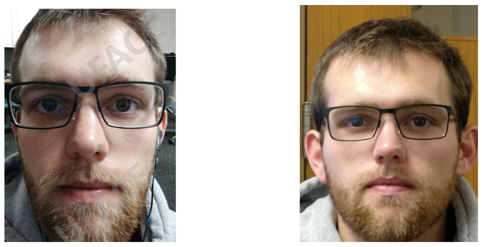
VIII. Security Analysis

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). As a result, our proposed protocol can be easily proved to be secure under the semi-honest model of SMC by using the simulation argument [12]. However, due to the space limit, we do not include the proof here. Instead, we will discuss the robustness of our UFace system against three common types of attacks:

Impersonation attack: This 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 [9]. 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 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 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 Figure 8). Our experiments with 20 users have proved that the LBP algorithm is capable of distinguishing these 2 types of images.



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

Fig. 8. Close-up Photos vs. Zoom-In Photos

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.

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.

TABLE I
EXPERIMENTAL SETTINGS

Parameters	**Values**
Number of users	**20**
Number of photos	**60**
Number of tests	**800**
Encryption key size	**1024 bits**
LBP threshold	**88.5%**
Grid size	1x1, 2x2, **4x4**, 8x8
Photo Size	128×128, **256×256**, 512×512, 1024×1024

IX. Experimental Study

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 which uses the Snapdragon 820 processor (4 cores: 2x2.15 GHz and 2x1.6 GHz) and contains 6 GB RAM. The web services, data server, and key server were all run on the same virutal machine that used an Intel Xeon processor (6 cores at 3.5 GHz) and had 8.5GB of RAM. All photos taken of users were using their own Android phones and the quality of the camera varied.

The performance of the UFace system is evaluated using two metrics: (i) *accuracy*, and (ii) *response time*. Table I summarizes the parameters tested. In what follows, we report the detailed experimental results.

A. Accuracy Analysis

The first round of experiments was aimed to identify the ideal picture size and grid size to achieve the highest accuracy rate. This was done by varying the grid size and keeping the picture size constant and then repeat this test by varying the picture size instead and keeping the grid size constant. The analysis was done by testing each of the user's 20 selfies against another selfie of the user and a zoomed in image. There were 2 tests for each user's data, so for 20 users that comes to be 2×20^2 or 800 different tests.

For varying the grid size, the general trend was that as the grid size increased, the similarity between 2 images decreased. This occurs because in small grid sizes, a pixel's label may match with a pixel in the opposite corner of the same section. As the grid size increases, the number of pixels in each section decreases so it's more difficult for the labels to match unless the images are similar. This maintains spatial information regarding each user's face. However, since it's incredibly difficult to take nearly identical photos, the grid size needs to not be extremly large for false negatives to occur.

Varying the picture size had an opposite trend: as picture size increased so did similarity between 2 images. This occurs because this increases the number of labels that can be placed into each bin for LBP. Since there are more labels in general, the number of labels that overlap will increase and thus the histogram comparison algorithm will generate a greater accuracy. However, if the picture size is too large, different users or zoomed-in images of the user will match correctly.

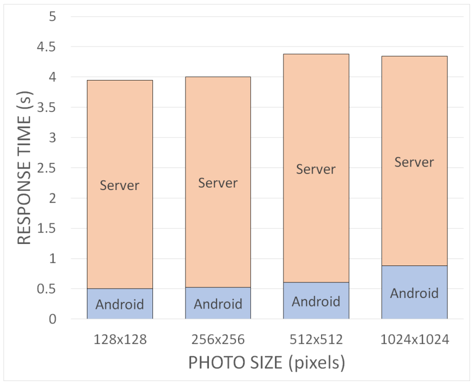


Fig. 9. Effect of photo size

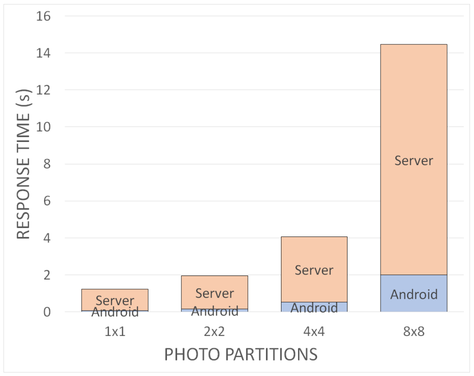


Fig. 10. Effect of Number of Photo Partitions

B. Time Analysis

The first analysis analyzed the accuracy of each paramter; however, time also needs to be accounted for to create an application that can done in a similar time to common password authenticaiton methods. It should be noted that the number of bits used per bin was set to 20 to account for all different number of pixels that could occur in a section instead of the ideal value for those settings. This will only slow the time down and results using the ideal values are faster.

The time taken to encrypt the feature vector on the phone and execute the garbled circuit are both directly related to grid size. The reason for this is because if the grid size increases, then the number of bins also increases. This means that there will need to be more encryptions to occur and the garbled circuit will need to grow in size to account for the extra bins to compare. Picture size does not vary the time since this just changes the value in each of the bins. The effects of varying the picture size can be seen in Figure 9.

However, the time taken to execute the LBP algorithm does vary with picture size. As the pixel count increases, LBP needs to be executed more often. In fact, the time taken to execute LBP quadrupled in each consecutive picture size because each test increased the pixel count by 4. Grid size did not vary the time since the grid size just changed where each pixel's label was stored. Figure 10 shows the effect of paritioning.

TABLE II
FACE RECOGNITION ACCURACY

Comparison Test	Pass Count	Fail Count
vs Self Close-Up	20/20	0/20
vs Self Zoomed-In	0/20	20/20
vs Others' Close-Up	0/19	19/19
vs Others' Zoomed-In	0/19	19/19

C. Ideal Parameters Analysis

The ideal values were determined to be a grid size of 4x4 and a picture size of 256x256 pixels. The final parameter not mentioned above is the threshold value which was analyzed with the previous 2 tests. A threshold value of .885 provided 100% user match for 2 close-up images and 100% fail for all other photos (zoomed-in photos or other users' close-up or zoomed-in photos). These values are highlighted in Table I. These ideal values were then saved and the entire test was run again using the same 60 photos.

Table II summarizes the authentication results of the 20 users close-up facial images against every other type of image. Here, "Pass" refers to that the photo passed the authentication, i.e., the matching score is above the 0.885 threshold; "Fail" refers to authentication failure, i.e., the access to the user account would be denied.

Each user's close-up facial image was tested against another close-up image and a zoomed-in image of the users face along with a close-up and zoomed-in image of every other user's face. The first 2 rows state that each user's stored close-up selfie matched to another close-up selfie of the same user while not matching to a image of the user's face that was zoomed-in. The final 2 rows state that each user's stored close-up selfie failed to match against any users' photos. 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 has differences from the zoom-in photos caused by the optical distortion due to the photos having different focal points. Therefore, the LBP algorithm can distinguish them.

X. CONCLUSION

In this paper, we present a privacy-preserving face authentication system, called UFace, for authenticating web services on a mobile device. 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. UFace successfully prevents the common threat from the impersonation attack by using users' online images. The UFace system has been implemented on Android phones for performance evaluation. The experimental results have demonstrated that the UFace system is capable of fulfilling the authentication task accurately and efficiently within seconds.

REFERENCES

[1] Total number of websites.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *Computer vision-eccv 2004*, pages 469–481. Springer, 2004.

[3] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[4] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp - a system for secure multi-party computation. In *Proceedings of the ACM Computer and Communications Security Conference (ACM CCS)*, October 2008.

[5] Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In *Computer Security–ESORICS 2011*, pages 190–209. Springer, 2011.

[6] J. Bringer, H. Chabanne, and A. Patey. Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends. *Signal Processing Magazine, IEEE*, 30(2):42–52, March 2013.

[7] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[8] Hu Chun, Yousef Elmehdwi, Feng Li, Prabir Bhattacharya, and Wei Jiang. 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.

[9] NM Duc and BQ Minh. Your face is not your password. In *Black Hat Conference*, volume 1, 2009.

[10] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, pages 235–253. Springer, 2009.

[11] David Evans, Yan Huang, Jonathan Katz, and Lior Malka. Efficient privacy-preserving biometric identification. In *Proceedings of the 17th conference Network and Distributed System Security Symposium, NDSS*, 2011.

[12] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter Encryption Schemes. Cambridge University Press, 2004.

[13] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *The 20th USENIX Security Symposium*, August 2011.

[14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 2007.

[15] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology–EUROCRYPT 2008*, pages 146–162. Springer, 2008.

[16] Yan Li, Yingjiu Li, Qiang Yan, Hancong Kong, and Robert H. Deng. 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*, CCS '15, pages 1558–1569, New York, NY, USA, 2015. ACM.

[17] Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. Scifi-a system for secure face identification. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 239–254. IEEE, 2010.

[18] P. Paillier. 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. Springer-Verlag.

[19] Doyel Pal, Praveenkumar Khethavath, Johnson P Thomas, and Tingting Chen. Secure and privacy preserving biometric authentication using watermarking technique. In *Security in Computing and Communications*, pages 146–156. Springer, 2015.

[20] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology–ICISC 2009*, pages 229–244. Springer, 2010.

[21] Jaroslav Sedenka, Sathya Govindarajan, Paolo Gasti, and Kiran S Balagani. Secure outsourced biometric authentication with performance evaluation on smartphones. *Information Forensics and Security, IEEE Transactions on*, 10(2):384–396, 2015.

[22] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[23] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography*, pages 457–473. Springer, 2009.

[24] Anurag Tagat. Online fraud: too many accounts, too few passwords.

[25] Xiaoyang Tan and Bill Triggs. Enhanced local texture feature sets for face recognition under difficult lighting conditions. *Image Processing, IEEE Transactions on*, 19(6):1635–1650, 2010.

[26] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[27] Yi Xu, True Price, Jan-Michael Frahm, and Fabian Monrose. Virtual u: Defeating face liveness detection by building virtual models from your public photos. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 497–512, 2016.

[28] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.