Computer Communications 180 (2021) 97–108

Contents lists available at ScienceDirect

Computer Communications

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

Towards real-time privacy-preserving video surveillance[☆]

Elmahdi Bentafat, M. Mazhar Rathore, Spiridon Bakiras[*]

Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar

A R T I C L E I N F O

Keywords:
Video surveillance systems
Privacy
Face recognition
License plate recognition
Homomorphic encryption

A B S T R A C T

Video surveillance on a massive scale can be a vital tool for law enforcement agencies. To mitigate the serious privacy concerns of wide-scale video surveillance, researchers have designed secure and privacy-preserving protocols that obliviously match live feeds against a suspects' database. However, existing approaches are very expensive in terms of computation and communication costs and, as a result, they do not scale well for ubiquitous deployment. To this end, we propose a general framework for privacy-preserving identification that operates by storing an encrypted version of the suspects' database at the video cameras. We show that this approach (i) reduces the protocol to a single round of communication between the camera and the server and (ii) speeds up the computation times significantly through the use of input-independent precomputations. We apply our framework to two practical use-cases, namely, face and license plate number recognition. In addition to the identification result, our face recognition protocol discloses some trivial information to the database server; however, this information is not sufficient for the server to infer any meaningful characteristics about the underlying individuals. On the other hand, the license plate recognition protocol is provably secure and can also handle minor character recognition errors that often occur in such systems. We implemented working prototypes of both surveillance systems and our experimental results are very promising. In the case of face recognition, and for a database of 100 suspects, the online computation time at the camera and the server is 155 ms and 34 ms, respectively, while the online communication cost is only 12 KB. Similarly, for a database of 3000 entries, license plate recognition requires only 232 ms and 75 ms at the camera and the server, respectively, while the online communication cost is 375 KB.

1. Introduction

Surveillance systems are being deployed in numerous countries around the world. An effective video surveillance system automatically monitors all available data feeds, extracts feature vectors from the individual targets, compares them against a suspects' database, and raises an alarm when a match is found. Nevertheless, this approach raises significant privacy concerns, because all individuals with known feature vectors can be tracked on a daily basis. Analyzing such information-rich datasets has the potential to reveal sensitive personal information, including home and work locations, health issues, religious affiliations, etc. Even if we trust the law enforcement authorities to protect the location privacy of their citizens, the stored location data may still be accessed by malicious users, such as rogue insiders or hackers.

As a result, the research community has proposed several methods that provide privacy-preserving systems for face recognition [1–3] and license plate matching [4,5]. In particular, these methods execute a secure two-party protocol between the camera and the database server, which lets the camera learn in zero-knowledge whether a captured feature vector matches one of the suspects in the database. If a match is found, the id number of the suspect is revealed; otherwise, the protocol discloses no information to either party. These protocols first compute an encrypted similarity score (Euclidean or Hamming distance) for each suspect in the database, using an additively homomorphic cryptosystem. Then, a variety of techniques are employed to identify the matching suspect, if and only if the underlying similarity distance is below a certain threshold. These techniques involve standard cryptographic primitives for secure computations, such as homomorphic encryption, garbled circuits, and oblivious transfer.

Nevertheless, all the aforementioned systems suffer from high computational and communication costs that render them impractical for wide-scale deployment. For instance, the Eigenfaces implementation by Sadeghi et al. [3] for face recognition necessitates 40 s of online computations to match a single face against a database of 320 suspects. In addition, the online communication cost is over 5 MB. Similarly, SCiFI [2] reports 31 s of online computations for a database of 100 suspects. Another significant limitation of these protocols is their reliance

☆ A preliminary version of this paper appeared in the Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS 2020).

* Corresponding author.

E-mail addresses: ebentafat@hbku.edu.qa (E. Bentafat), mrathore@hbku.edu.qa (M.M. Rathore), sbakiras@hbku.edu.qa (S. Bakiras).

https://doi.org/10.1016/j.comcom.2021.09.009
Received 12 February 2021; Received in revised form 13 July 2021; Accepted 9 September 2021
Available online 17 September 2021

on offline computation and communication that has to be performed for every face that is captured by the camera. While the offline tasks reduce the overall online cost dramatically, it is not feasible to process and store the underlying data for potentially millions of detected faces on a daily basis.

To this end, our work introduces the first practical system for privacy-preserving video surveillance on a large scale. In particular, we propose a general framework that applies to different surveillance applications, such as face recognition and license plate matching. The efficiency of our approach stems mainly from two design decisions. First, the suspects' database is distributed to all cameras, after it is encrypted with the public key of the law enforcement agency. As such, the expensive operations for computing the encrypted similarity scores are performed at the surveillance cameras, thus alleviating the server's computational load. The local database copy also allows the cameras to precompute most values that are involved in the encrypted distance computations. More importantly, unlike existing approaches, the offline computations are performed only once, during the system's initialization. In addition to precomputations, our methods employ an efficient elliptic curve cryptosystem (ElGamal [6]) that reduces significantly the computational and communication costs.

Our second decision is to disclose to the server a random permutation of *obfuscated* similarity scores between the captured feature vector and all suspects in the database. The server then decrypts all the similarity scores and, based on the plaintext values, it is able to deduce whether one of them is a potential match. As a result, the protocol involves a single round of communication for the server to learn the (binary) result of the identification. If a match is found, an additional verification protocol is invoked, where the server receives the suspect's id and optionally receives the image of the potential suspect.

In our previous work [7], we applied this framework to the case of privacy-preserving face recognition. Given the wide range of the underlying similarity scores, our obfuscation step for this use-case does not result in a zero-knowledge identification. Nevertheless, we have showed that, the leaked information is not sufficient for the server to infer any meaningful characteristics regarding the target's feature vector. We built the corresponding video surveillance system on top of the OpenFace [8] platform that implements the face recognition layer. OpenFace is one of the most accurate open-source face recognition systems that employs Google's FaceNet [9] algorithm. Besides its high accuracy, a notable advantage of OpenFace over other approaches is its compact feature vector that speeds up considerably the encrypted distance computations. We demonstrated experimentally that our system reduces the costs by orders of magnitude compared to the current state-of-the-art approaches.

In this paper, we extend our previous work in two directions. First, we improved the face recognition implementation of our system, by incorporating several code optimizations. As a result, we were able to achieve a higher frame per second rate for the live video feed. Second, we extended our framework to the case of privacy-preserving license plate recognition. The proposed protocol is provably secure and it is also able to handle minor character recognition errors that may be caused by the poor quality of the captured image. To the best of our knowledge, this is the first method in the literature that addresses this issue. We built the corresponding surveillance system on top of the OpenALPR [10] platform that leverages deep learning techniques to efficiently extract license plate numbers from images or video sequences. As in the case of face recognition, we demonstrate that our system is orders of magnitude faster than existing methods.

The rest of the paper is organized as follows. Section 2 presents a literature review of privacy-preserving face recognition and license plate recognition systems, and Section 3 discusses the main tools that we utilized in our implementation. Section 4 introduces our problem definition and describes the underlying threat model. Section 5 discusses our general framework for privacy-preserving video surveillance, while Sections 6 and 7 describe the details of the two use-cases. Section 8 evaluates the security of our systems, and Section 9 presents the implementation details and summarizes our experimental results. Finally, Section 10 concludes our work.

2. Related work

2.1. Privacy-preserving face recognition

The first privacy-preserving face recognition protocol is due to Erkin et al. [1] in 2009. It leverages the Eigenfaces algorithm for face recognition, but is very inefficient in terms of online performance. Specifically, the protocol requires $O(\log M)$ rounds of online communication (M is the number of suspects in the database) and heavy public key homomorphic operations over the ciphertexts. Sadeghi et al. [3] improved the performance of Erkin's work by shifting some computations into a precomputation phase, and using garbled circuits [11] to compute the *Minimum* function. In a recent work, Xiang et al. [12] further improved upon the aforementioned protocols [1,3] by outsourcing the expensive server computations to the cloud.

SCiFI [2] is the only protocol in the literature that is not based on the Eigenfaces representation. Instead, the authors proposed a novel face recognition method that takes into account the appearance of certain facial features. In SCiFI, each face is represented with a 900-bit vector, while the similarity score is simply the Hamming distance between two vectors. After the Hamming distance is computed, the result of the suspect identification is revealed through a 1-out-of-d_{max}+1 oblivious transfer protocol [13], where d_{max} is the maximum theoretical Hamming distance. One advantage of this approach is that Hamming distance computations on the ciphertext space are significantly faster than the Euclidean ones. Finally, various studies have used similar cryptographic tools, mainly garbled circuits and oblivious transfer, in the context of biometric identification. In particular, researchers have proposed several efficient protocols to compute the similarity scores, including Hamming distance, Euclidean distance, Mahalanobis distance, and scalar product [14–18].

2.2. Privacy-preserving license plate recognition

Sunil et al. [4] introduced the first protocol for privately matching Dutch car license plate numbers. The authors proposed a simple and accurate character recognition algorithm where the resulting feature vectors were converted into integer numbers. They considered two different encryption schemes in their implementation, namely, Gentry's fully homomorphic cryptosystem [19] and Paillier's additively homomorphic cryptosystem [20]. Even though Paillier's scheme is considerably faster, their experimental results show that the overhead incurred by the cryptographic layer is still very high for a wide-scale adoption.

Vaishnav et al. [5] replaced the Paillier implementation of the aforementioned protocol with an optimized version, and they were able to reduce the computational cost by a factor of 10. Specifically, they showed that matching a license plate against a database of 2500 entries takes 7 seconds, using a 2048-bit RSA modulus as the Paillier key. In a more recent work, the same authors [21] improved their previous work, by leveraging a lightweight cryptosystem called HE1N [22], which is a symmetric, integer-based, fully homomorphic encryption scheme. Nevertheless this protocol is most likely insecure, because symmetric homomorphic encryption is not widely accepted by the crypto community. For example, a similar cryptosystem by Trostle and Parrish [23] has been completely broken by lattice-based attacks, such as the Lenstra–Lenstra–Lovász (LLL) reduction [24].

3. Tools

3.1. OpenFace

OpenFace [8] is an open-source face verification and recognition system that maps face images to a compact Euclidean space. It is a deep convolutional network trained method for face recognition that achieves an accuracy of 92.95% on the Labeled Faces in the Wild (LFW) benchmark, one of the largest publicly-available datasets. OpenFace matches very well the performance of FaceNet [9] and DeepFace [25], despite the small size of the trained network. The advantage of OpenFace is the face representation efficiency that consists of 128 features. The similarity score between two faces is represented by the Euclidean distance of the two feature vectors, and ranges between 0 (for the same image) and 4. A threshold $t = 0.9$ has been set empirically by the system developers of OpenFace, such that a distance less than t indicates a positive match.

3.2. OpenALPR

OpenALPR [10] is an open-source Automatic License Plate Recognition (ALPR) library. It is developed in C/C++ and has bindings in Python, Java, and C#. The library was published as open-source software in late 2015, but is also available as a commercial product that provides extended functionalities, such as a video stream analyzer and a cloud-based API. License plate recognition in OpenALPR consists of multiple steps. First, the detection engine detects potential license plate regions on a given image, while the binarization process converts these regions into black and white. Next, a character analysis algorithm identifies the blobs in the plate region, which is followed by a process that detects the edges of the plate number. Then, the deskew algorithm transforms the detected image to an ideal size, and the character segmentation engine isolates the individual characters. Finally, an optical character recognition (OCR) algorithm recognizes these characters along with their confidences, and a post-processing step generates a list of possible results based on these confidences. To perform these operations, OpenALPR relies on other libraries, such as OpenCV [26] for image processing based on deep learning frameworks, and Tesseract OCR [27] for optical character recognition. Notice that, OpenALPR is a deep neural network-based method, and relies on models trained over large datasets, in order to reach a high accuracy.

3.3. Homomorphic encryption

Homomorphic cryptosystems [28] allow for the evaluation of certain arithmetic operations directly on the ciphertext domain. Fully homomorphic encryption (FHE) [19] supports both addition and multiplication operations and can, thus, be used to evaluate any circuit over encrypted data. Nevertheless, FHE schemes are still very inefficient to be used in real-time applications, such as video surveillance. Instead, similar to previous work, we built our protocol on top of additively homomorphic cryptosystems, such as Paillier [20] or ElGamal. More specifically, we opted for an implementation of ElGamal's cryptosystem over elliptic curves, due to its computational efficiency and compact ciphertexts (128 bytes). The cryptosystem consists of the following functions:

- **Key generation**: Instantiate an elliptic curve group of prime order q with generator P. Choose a *private* key x uniformly at random from $\mathbb{Z}_q^*$ and set the *public* key $Q = x \cdot P$.
- **Encrypt**: Let m be the secret message. Choose r uniformly at random from $\mathbb{Z}_q^*$ and compute ciphertext $\mathsf{Enc}(m) = \langle r \cdot P, (m+r) \cdot Q \rangle$.
- **Decrypt**: Compute $m \cdot Q = (m+r) \cdot Q - x \cdot r \cdot P$ and solve the discrete log to recover m.

ElGamal's scheme is semantically secure and its security is based on the decisional Diffie–Hellman assumption. Note that, in our implementation, we utilized a look-up table of precomputed $m \cdot Q$ values (for all theoretically possible values of m) in order to speed up the discrete log computations at the database server.

The homomorphic properties of ElGamal's cryptosystem over elliptic curves are shown below.

- **Homomorphic addition**: Given the encryption of two messages m_1 and m_2, $\mathsf{Enc}(m_1) + \mathsf{Enc}(m_2)$ is equal to

$$\langle r_1 \cdot P, (m_1 + r_1) \cdot Q \rangle + \langle r_2 \cdot P, (m_2 + r_2) \cdot Q \rangle =$$

$$\langle (r_1 + r_2) \cdot P, (m_1 + m_2 + r_1 + r_2) \cdot Q \rangle = \mathsf{Enc}(m_1 + m_2)$$

- **Homomorphic scalar multiplication**: Given a plaintext scalar λ and the encryption of a message m, $\lambda \cdot \mathsf{Enc}(m)$ is equal to

$$\langle \lambda \cdot r \cdot P, (\lambda \cdot m + \lambda \cdot r) \cdot Q \rangle = \mathsf{Enc}(\lambda \cdot m)$$

4. Security definition and threat model

We assume a wide-scale surveillance environment, where a large number of cameras, equipped with moderate computational, storage, and communication capabilities, are deployed throughout a city. The database server (law enforcement) holds a database $\mathbb{S} = \{S_1, S_2, \dots, S_M\}$ of M suspects, where each suspect S_i is represented by an Nth dimensional feature vector $\mathbf{x}_i$. More specifically, for face recognition, the feature vectors are generated from OpenFace's deep learning model and consist of $N = 128$ values. For license plate matching, the feature vectors are extracted using OpenALPR with $N = 8$ (max number of characters on a license plate). During the system initialization, the database server shares an encrypted version of $\mathbb{S}$ (to be discussed later) with all cameras, using its own public key Q that is also known to all cameras. Every camera will then capture all passing-by faces/vehicles and, for each candidate C_j, compute its feature vector $\mathbf{y}_j$ using OpenFace/OpenALPR. What follows, is a two-party protocol between the camera and the database server, where

- The server outputs a random permutation π_j of obfuscated similarity scores between $\mathbf{y}_j$ and $\mathbf{x}_i, \forall i \in \{1, 2, \dots, M\}$.
- The camera has no output.

When the protocol's output is revealed to the server, it will immediately disclose the identification result. In particular, for face recognition, a positive match is signified by a negative similarity score, whereas in the case of license plate recognition, a match is triggered by a similarity score of zero. Note, however, that the identity of the suspect is still unknown due to the underlying permutation. In that case, the camera and the server invoke a separate two-party protocol, where the camera verifies that the similarity score is indeed negative/zero. During that protocol,

- The server receives the actual id of the matching suspect and (optionally) receives the captured image from the camera.
- The camera receives the actual similarity score and id of the matching suspect.

We assume that the server and all cameras are semi-honest players. In other words, they will follow the protocols correctly, but try to infer some non-trivial information about the other party's input from the communication transcript. For example, the camera might want to learn the plaintext content of the suspects' database, while the server might want to infer some information about the captured faces that do not produce a database match. We also allow the server to act maliciously after the initial identification result, by falsely claiming that a certain similarity score is negative/zero. Such behavior will be discovered during the subsequent verification protocol. Finally, we should note that our protocol cannot protect against illegitimate inputs from any of the parties. For instance, the server can insert into their database $\mathbb{S}$ an innocent civilian that it wants to track, while the camera can test whether a specific individual is part of $\mathbb{S}$ by using their feature vector in the identification protocol. However, none of the existing privacy-preserving protocols can protect against such attacks, since they are not cryptographic in nature.

5. Privacy-preserving surveillance framework

In this section, we introduce the different phases of our privacy-preserving surveillance framework. Nevertheless, each surveillance application has its own specificities, which will be explained in detail in the upcoming sections.

5.1. Offline phase

The server first instantiates an elliptic curve group of prime order q (as described in Section 3) and generates its public and private keys. The public key is distributed securely to all surveillance cameras in the city. Next, the server employs a recognition method to extract the feature vector $\mathbf{x}_i$ for every suspect $S_i \in \mathbb{S}$. The feature vectors are then encrypted with its public key and sent to the surveillance cameras. Finally, both the server and the individual cameras, precompute a series of public key operations that will be used to speed up the online surveillance process. It is worth noting that, unlike existing approaches, the offline costs of our framework are incurred only once and are independent of the number of objects that are captured by the camera.

5.2. Similarity score computation

Following the offline phase, our system is ready for real-time video surveillance. When a new candidate is identified, the camera generates its plaintext feature vector and, using the precomputed values from the offline phase, it evaluates the encrypted similarity score for every suspect in the database. The similarity metric will depend on the underlying surveillance application.

5.3. Similarity score obfuscation

Once the similarity scores are computed, the camera performs the similarity score obfuscation step. This is accomplished with an affine-like transformation, involving multiplication and addition operations. However, the transformation is performed in a way that it does not affect the matching decisions at the server. The obfuscated scores are then randomly permuted and forwarded to the database server.

5.4. Matching

The matching operation at the server is straightforward. It simply decrypts all ciphertexts and, based on the obfuscated scores, the server can determine whether a positive match has occurred. Note that, for efficiency, the decryption operations may also involve precomputed values from the offline phase. Upon signaling a positive match, a verification protocol is invoked in order for the server to learn the id of the potential suspect and (optionally) receive the surveillance image. This step is necessary to prevent a malicious server from requesting footage of random individuals that did not actually produce a database match.

6. Use-case 1: Face recognition

6.1. Offline phase

First, the server employs OpenFace to generate the feature vectors $\mathbf{x}_i$ for every suspect $S_i \in \mathbb{S}$. By default, OpenFace operates over floating point numbers, so we first had to convert the vectors into integers before applying any homomorphic operations. We empirically computed the normalization parameters for a floating point representation f as follows: $\lfloor f \times 400 + 128 \rfloor$, where $f \in \mathbb{Q} : -0.32 < f < 0.32$. With this transformation, every element in a feature vector is an integer in the range $[0, 256)$, thus allowing us to represent a vector with just 128 bytes. Furthermore, the transformation does not result in a significant loss of accuracy, as illustrated in Table 1. Specifically, the table depicts the accuracy results from various state-of-the-art face recognition algorithms, and also quantifies the loss of accuracy due to the normalization of the features values. For our system, we rerun the evaluation step of the CNN model using our normalization technique in order to use integers in the range $[0, 256)$ instead of floats and, out of 13,233 images, we had only one misidentification compared to the original OpenFace implementation. Note that FaceNet and DeepFace are more accurate than OpenFace because they are trained on much larger datasets.

Table 1
Accuracy results on the LFW benchmark [8].

Model	Accuracy
Human	97.53%
EigenFaces	60.02% ± 0.79
FaceNet	99.64% ± 0.9
DeepFace	97.35% ± 0.25
OpenFace	92.95% ± 1.34
OpenFace, normalized	92.92% ± 1.36

Given suspect S_i's feature vector $\mathbf{x}_i$ and a potential candidate's vector $\mathbf{y}_i$, the first step of OpenFace's face recognition algorithm is to compute the Euclidean distance between the two feature vectors. In the ciphertext domain, it is only feasible to compute the squared Euclidean distance, i.e.,

$$d_i^2 = \sum_{j=1}^{N} (x_{i,j} - y_{i,j})^2 = \sum_{j=1}^{N} (x_{i,j}^2 + y_{i,j}^2 - 2x_{i,j}y_{i,j}) \tag{1}$$

where $N = 128$ is the vector dimensionality. In the ciphertext domain over elliptic curves, this is equivalent to

$$\mathsf{Enc}(d_i^2) = \sum_{j=1}^{N} \mathsf{Enc}(x_{i,j}^2) + \sum_{j=1}^{N} \mathsf{Enc}(y_{i,j}^2) + \sum_{j=1}^{N} y_{i,j} \cdot \mathsf{Enc}(-2x_{i,j}) \tag{2}$$

Therefore, for the cameras to correctly compute $\mathsf{Enc}(d_i^2)$, the server will send them an encrypted version of the database $\mathbb{S}$, consisting of

- $\sum_{j=1}^{N} \mathsf{Enc}(x_{i,j}^2), \forall i \in \{1, 2, \ldots, M\}$.
- $\mathsf{Enc}(-2x_{i,j}), \forall i \in \{1, 2, \ldots, M\}, j \in \{1, 2, \ldots, N\}$.

As such, the offline communication cost of our protocol is $(N+1) \times M \times T$ bytes, where T is the size of an ElGamal ciphertext. Due to the semantic security of the cryptosystem, the cameras cannot infer any information regarding the feature vectors of the suspects.

After a camera receives the encrypted database, it performs a series of offline precomputations, in order to speed up the online computation of the similarity scores. In particular, the camera will precompute all possible values for the second and third terms of Eq. (2), which is feasible due to the limited range of $y_{i,j}$ (just 256 distinct values). The computational cost involves $256 \times N \times M$ elliptic curve point multiplications and 256 encryption operations. Additionally, the storage requirements at the camera (for the database and all precomputed values) is $(256 + M + 256 \times N \times M) \times T$ bytes. Even for large databases (e.g., $M = 1000$), the storage cost is approximately 4 GB, which is very reasonable for a low-cost camera.

Nevertheless, if a camera does not possess the storage capacity to hold the entire set of precomputed values, we may still gain a lot in performance if we store partial information. (We will illustrate this in our experimental results.) As shown in Fig. 1, the coefficients of a feature vector are not uniformly distributed over the entire range, but instead, values ranging from 64 to 191 tend to occur more frequently. As such, the camera may only precompute, say, 50% of the values (for $y_{i,j} \in [64, 191]$) and perform the remaining elliptic point multiplications, i.e., $y_{i,j} \cdot \mathsf{Enc}(-2x_{i,j})$, on the spot.

At the server side, the offline cost to compute the encrypted database is $2 \times N \times M$ encryption operations plus $N \times M$ elliptic curve point additions, which is trivial for a powerful multi-core server. On the other hand, in order to speed up the decryption operations that

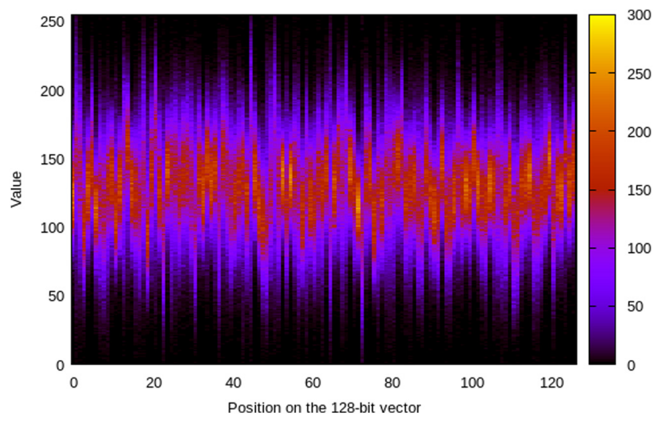


Fig. 1. Heat map of feature vector coefficients for 13,095 faces.

constitute the bottleneck of the online identification protocol, we need to precompute a large number of elliptic curve points (lookup table), as explained in Section 3. Assuming a maximum bit-length of k bits for the obfuscated similarity scores, the server will precompute and store 2^k 32-byte values with a computational cost of 2^k additions (that are relatively cheap). In our system, the lookup table was implemented as a hash table with open addressing, using buckets of size two. This generated a hash table with a load factor of 0.5. We set $k = 30$, and opted to store only 128 bits of the key value instead the entire 256 bits, which reduces the size of the lookup table to 34 GB.

6.2. Similarity score computation

Each camera captures all passing-by faces and, for each face C_j, it generates the plaintext feature vector $\mathbf{y}_j$. Based on the generated vector elements, the camera selects the corresponding ciphertexts from the precomputed values and evaluates the encrypted squared Euclidean distances $\mathsf{Enc}(d_i^2)$ for every suspect i, as given in Eq. (2). This task entails, for all M suspects, $(2\times N+1)\times M$ elliptic curve point additions. Each distance is then adjusted by subtracting the normalized similarity threshold t, thus generating an encrypted similarity score s_i that is (i) positive for a non-match or (ii) negative for a match. Therefore, the encrypted similarity score for suspect i is computed as

$$\mathsf{Enc}(s_i) = \mathsf{Enc}(d_i^2 - t) = \mathsf{Enc}(d_i^2) + \mathsf{Enc}(-t) \tag{3}$$

By precomputing the encrypted threshold value (constant), the computational cost of this step is M point additions. To summarize, the overall cost for computing the similarity score is $2\times(N+1)\times M$ point additions. Finally, we should mention that, based on the normalization parameters given is Section 5.1, the normalized similarity threshold is set to $t = (0.9\times 400)^2 = 129,600$.

6.3. Similarity score obfuscation

For every score s_i, the camera selects two uniformly random numbers $r_1, r_2 \in (0, 2^\ell)$ and masks the score as

$$\mathsf{Enc}(\delta_i) = r_1 \cdot \mathsf{Enc}(s_i) + \mathsf{Enc}(r_2) \tag{4}$$

In order to avoid reversing the sign of the similarity score, we always choose $r_1 > r_2$. Note that we may precompute some encryptions of r_2 and reduce the computational cost of this step to M point multiplications and M additions only. The exact value of ℓ depends on the memory specifications of the server, and affects both the security and the performance of our scheme. For this application, we empirically determined the max value for the similarity score to be $< 2^{19}$ and set $\ell = 11$, which limits the obfuscated scores to values $< 2^{30}$. Note that, since $r_2 < 2048$, we may precompute all possible encryptions of r_2 and

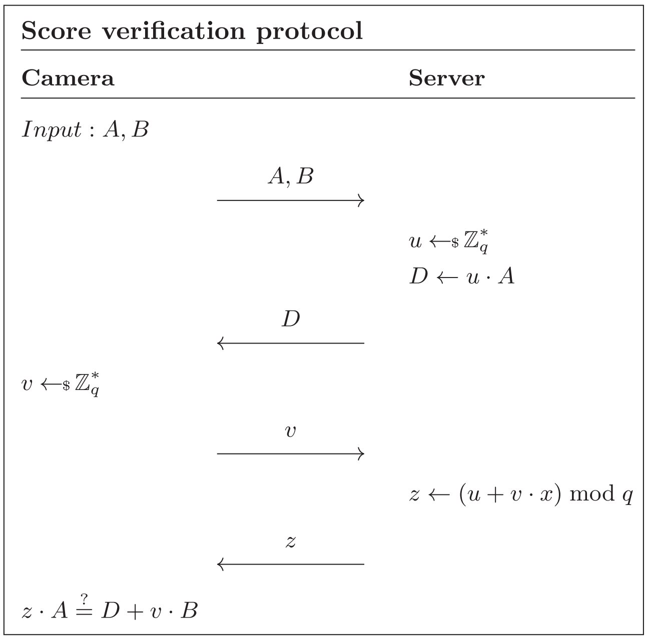

Score verification protocol
Camera
Server
$Input : A, B$
A, B
$u \xleftarrow{\$} \mathbb{Z}_q^*$
$D \leftarrow u \cdot A$
D
$v \xleftarrow{\$} \mathbb{Z}_q^*$
v
$z \leftarrow (u + v \cdot x) \bmod q$
z
$z \cdot A \overset{?}{=} D + v \cdot B$

Fig. 2. Score verification protocol.

reduce the computational cost of this step to M point multiplications and M point additions. Also note that the multiplication operations are quite cheap, as they involve ℓ-bit scalars.

6.4. Matching

The matching operation is performed exactly as explained in Section 5.4. Under normal conditions, the overwhelming majority of captured faces will not produce a database match, so the cost of the matching protocol is dominated by the M decryption operations, each requiring one point multiplication and one point addition (due to the stored lookup table). The communication cost involves the transmission of M ciphertexts and is, thus, equal to $M \times T$ bytes.

In the rare case of a positive match, the verification protocol is invoked as follows. The server first informs the camera of the suspect's position and score on the permuted vector, and the camera then looks up the suspect's real id and encrypted score in the permutation π that is temporarily stored in its local storage. Assume that the stored copy of the encrypted score is equal to $\mathsf{Enc}(s) = \langle r_1 \cdot P, (s + r_1)\cdot Q\rangle$. The camera will then generate encryption of the score s' that the server claims to be true: $\mathsf{Enc}(s') = \langle r_2 \cdot P, (s' + r_2)\cdot Q\rangle$. If $s = s'$, then a subtraction of the two ciphertexts will produce the encryption of the value zero. Therefore, the camera will compute

$$\mathsf{Enc}(s) - \mathsf{Enc}(s') = \langle (r_1 - r_2)\cdot P, (s - s' + r_1 - r_2)\cdot Q\rangle \tag{5}$$

which is supposedly equal to $\langle A, B\rangle = \langle r\cdot P, r\cdot Q\rangle$, for some unknown random r. As such, it suffices to prove that $x \cdot A = B$, where x is the server's private key. Essentially, the server has to prove to the camera that it knows the value x that satisfies the above equation. This is trivially done with Schnorr's identification protocol [29], as shown in Fig. 2. Initially, the server generates a uniformly random number u and computes D, which represents the server's commitment in the protocol. On the other hand, v is the challenge posed by the camera. The server's response z can only be computed by the party who knows x, and the camera accepts the result if and only if the last equation holds.

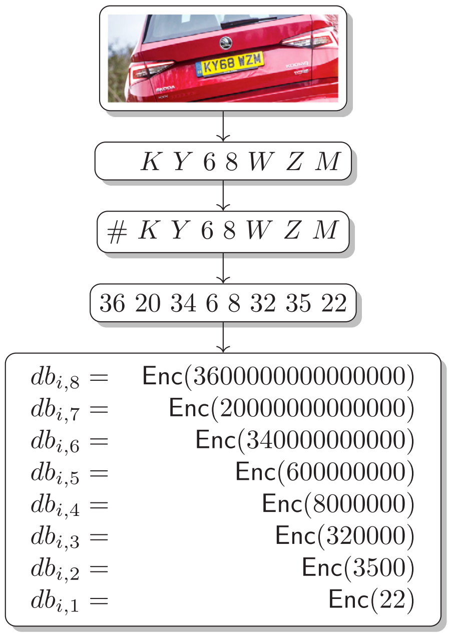

KY68 WZM
$K\ Y\ 6\ 8\ W\ Z\ M$
$\#\ K\ Y\ 6\ 8\ W\ Z\ M$
36 20 34 6 8 32 35 22
$db_{i,8} =$ Enc(3600000000000000)
$db_{i,7} =$ Enc(20000000000000)
$db_{i,6} =$ Enc(340000000000)
$db_{i,5} =$ Enc(600000000)
$db_{i,4} =$ Enc(8000000)
$db_{i,3} =$ Enc(320000)
$db_{i,2} =$ Enc(3500)
$db_{i,1} =$ Enc(22)

Fig. 3. Database encryption process.

7. Use-case 2: License plate recognition

7.1. Offline phase

Similar to face recognition, the offline phase involves (i) the generation of the encrypted suspects' database and (ii) the necessary precomputations that will be employed during the online phase. To derive the encrypted database, the server starts by encoding the license plate numbers stored in a text file. Specifically, given an alphabet $\mathcal{A}$ of size $|\mathcal{A}|$, we assign to each character an integer value from 0 to $|\mathcal{A}|-1$. We assume that the maximum number of characters on a license plate is N and, to accommodate plates with less than N characters, we introduce an additional *padding* character '#' in the alphabet. Padding is applied at the left hand side of the plate number. Most license plate numbers around the world consist of alphanumeric characters, i.e., the alphabet size would typically be equal to 37 (including the padding character).

After the characters are encoded into unique integers of size $\ell = \log|\mathcal{A}|$, each character is shifted to the left according to its location on the license plate, i.e., the character at position i is shifted bitwise by $(i-1)\cdot\ell$ positions, $\forall i \in \{1,2,\dots,N\}$. Finally, the server encrypts these values with its public key and distributes them to the surveillance cameras. The offline communication cost for the entire database $\mathbb{S}$ containing M license plate numbers is, therefore, $N \times M \times T$ bytes. The complete database encryption process in depicted in Fig. 3, for an alphabet of size $|\mathcal{A}| = 37$. At this point, the offline phase for the database server is complete. Unlike the use-case of face recognition, the server does not need to precompute any values for the decryption process since, as we will show later, the only plaintext value of interest is '0.' As such, the offline computation cost at the server is just $N \times M$ encryption operations.

Having the encrypted characters from a license plate number i, the camera can then combine them to construct the corresponding one-dimensional feature vector:

$$\mathbf{x}_i = \left[\sum_{k=1}^{N} db_{i,k}\right] \tag{6}$$

where $db_{i,k}$ represents the encryption of the shifted number at position k in the license plate of suspect i, as shown in Fig. 3. In other words, to compare two license plate numbers, we simply subtract (homomorphically) their feature vectors. If there is a match, the resulting ciphertext is an encryption of '0.' Nevertheless, optical character recognition systems, including OpenALPR's, are not perfect and may produce small errors during the recognition process. These can be caused by the internal image processing algorithms and/or the camera's image resolution. Such recognition errors will result in false negatives that fail to identify the corresponding suspect.

Therefore, in this work, we decided to address character recognition errors, in order to reduce the false negative rate of our system. Note that, false positives are not a major concern, because the error will be detected when the law enforcement receives the image of the suspect's license plate. Let us consider off-by-one errors first, i.e., out of the N possible characters, at most one will be recognized incorrectly. (We assume, however, that the total number of characters on the license plate is recognized correctly.) To handle such cases, the camera will compute an N-dimensional feature vector $\mathbf{x}_i$, where feature $x_{i,j}$ is computed as follows:

$$x_{i,j} = \sum_{k=1,k\neq j}^{N} db_{i,k} \tag{7}$$

In other words, each element j of the feature vector corresponds to the encryption of the license plate that does not include the jth character.

It is worth noting that, we can extend this approach to address l-character errors, i.e., the camera can compute an $\binom{N}{l}$-dimensional feature vector where each element is the encryption of the license plate that is missing l out of N characters. However, this method is not recommended for $l > 1$, because (i) it is expensive in terms of computation, storage, and communication costs and (ii) it may increase the number of false positives, thus negatively affecting the privacy of the underlying individuals. As a result, our system offers two variants of the license plate recognition protocol: (i) a basic version that lacks fault tolerance ($l = 0$), and (ii) an enhanced version that handles simple off-by-one errors ($l = 1$). Each camera will have the ability to configure this option, e.g., based on the underlying image resolution.

In terms of precomputations, each camera must compute and store the N different ciphertexts (one for each location) for every alphabet character. As such, when a new license plate is captured, the feature vector can be computed with cheap elliptic curve addition operations. Therefore, the offline computation cost at the camera is $N^2 \times M$ addition operations for the computation of the database feature vectors, and $|\mathcal{A}| \times N$ encryption operations for the necessary precomputations. Finally, the storage cost at the camera is quite small and consists of $(|\mathcal{A}| + M) \times N \times T$ bytes.

7.2. Similarity score computation

During real-time video surveillance, the camera identifies the license plates from passing-by vehicles. For each vehicle j, it extracts the license plate number C_j, and generates the (encrypted) feature vector $\mathbf{y}_j$. This is done by homomorphically adding a number of precomputed values from the offline phase. Then, for each suspect plate i in the database, it computes the encrypted similarity score vector $\mathbf{s}_i$ via element-wise subtraction as follows:

$$\mathbf{s}_i = \mathbf{x}_i - \mathbf{y}_j \tag{8}$$

Note that, to avoid performing the negation operation (which is expensive), we can simply precompute the encryptions of negated character values during the offline phase. For example, given the license plate character corresponding to value 15, the cameras will precompute the encryptions of −15, −1500, −150000, etc., instead of their positive counterparts. After all these optimizations, the online computation cost at the cameras is reduced to $N \times (N + M)$ point addition operations.

7.3. Similarity score obfuscation

Given that a positive match is signified by a zero similarity score, the obfuscation phase simply involves the multiplication of the score with a uniformly random $r \in [1, q)$. Unlike the use-case of face recognition, this obfuscation method gives us information-theoretic security since, for every possible real score value, we can find a random $r < q$ that produces the given obfuscated score. Not that, for computationally bounded adversaries, even the decryption of the obfuscated score is infeasible (for non-zero plaintexts) due to the discrete log nature of the cryptosystem. Nevertheless, these strong security guarantees come at a computational cost of $N \times M$ point multiplication operations with scalars of size $\log q$ bits.

7.4. Matching

The inherent security of the score obfuscation process allows the camera to send the $N \times M$ scores to the server in a sequential manner, i.e., without applying a random permutation. Therefore, the server can identify immediately a suspect license plate without contacting the remote camera. However, false positives are still possible, so the server may request an image of the captured license plate to confirm the match. In this case, the server and the camera will run the score verification protocol described in Fig. 2, where the server proves to the camera that a certain ciphertext is an encryption of ‘0.’ Regarding the matching operation, an encryption of ‘0’ will consist of a tuple $\langle r \cdot G, r \cdot P \rangle$ for some random integer $r < q$. Thus, for every received ciphertext, the server will multiply the first term with its secret key x and check whether the result is equal to the second term. Consequently, the computational cost for matching one captured license plate is $N \times M$ scalar-point multiplication operations.

8. Security

In this section, we analyze the security of the two different use-cases. For face recognition, we have employed several optimizations that relax the stringent privacy requirements of existing approaches, in order to make real-time video surveillance possible. These include (i) a single-round protocol that reveals a permuted list of obfuscated similarity scores and (ii) the use of a discrete log based cryptosystem that limits the degree of obfuscation that we can enforce. As a result, we cannot formally prove the protocol’s security, but instead provide empirical evidence that illustrate the difficulty of deriving any non-trivial information about the captured faces. On the other hand, license plate recognition is a considerably easier problem and we will prove its security under the simulation paradigm.

8.1. Face recognition system

We consider two types of attacks against our system. The first one is a complete privacy break, where the server is able to retrieve the plaintext version of the feature vector for some captured face. This is only possible if the server is able to correctly inverse the camera’s permutation and obfuscation steps and solve the underlying non-linear equations with N unknowns (assuming $M \geq N$). Nevertheless, this is infeasible due to (i) the exponential number $M!$ of possible permutation outcomes and (ii) the unpredictability of OpenFace’s deep learning approach to feature vector generation that makes it very difficult to link a similarity score to a specific face-suspect pair.

To illustrate the second point above, we analyzed the similarity scores generated by our system for four random faces from the LFW dataset. We selected 500 images from person P_1 and computed the (non-obfuscated) similarity scores against one image of P_1, P_2, P_3, P_4. The results are shown in Fig. 4, where it is evident that the obtained scores follow a Gaussian-like distribution with a large overlap among the different faces. In particular, for the non-matching faces, the large majority of similarity scores lie within the interval $[50K, 200K]$, thus preventing the server from inferring any non-trivial information about the underlying permutation.

The second type of attack is less severe and pertains to the ability of the database server to distinguish an unknown individual across multiple cameras. For example, suppose that a captured face generates an identical feature vector across a series of cameras. While the probability of that event is negligible, it is worth investigating the effect of the obfuscation step on the generated score distribution. Fig. 5 depicts the probability distribution of the obfuscated score bit-lengths against a database of 1000 suspects. P_1 is indistinguishable across two different obfuscations (for an identical feature vector) and all four distributions are very similar to each other with large overlaps. Note that, we are not interested in the distribution of negative scores, since a match will trigger the verification protocol that reveals the suspect’s identity.

8.2. License plate recognition system

In secure multiparty computation protocols, a straightforward approach to proving the security of a protocol is the simulation paradigm [30]. Specifically, it is sufficient to show that, for each party, we can simulate the distribution of the messages that it receives based only on that party’s input and output to the multiparty protocol. This is true because, if we can simulate each party’s view from just their input and output, then the messages themselves cannot possibly disclose any additional information.

Starting with the server, its input is the encrypted suspects’ database and the output can be one of the following: (i) the license plate does not match any suspect, (ii) the license plate is a 100% match to a suspect, and (iii) the license plate is a match to a suspect, but with an off-by-one error. These are easily simulated by sending $N \times M$ encryptions of random values $r < q$ for case (i), N encryptions of ‘0’ and $N \times (M - 1)$ random encryptions for case (ii), and one encryption of ‘0’ and $N \times M - 1$ random encryptions for case (iii). For the individual cameras, the input is a captured license plate and there is no output. Furthermore, the only messages that a camera receives are the ciphertexts of the encrypted database. As such, the simulator can simply generate $N \times M$ random encryptions because, given the semantic security of ElGamal’s cryptosystem, the camera cannot distinguish these ciphertexts from the ones that are produced by the server’s real input.

9. Experimental results

9.1. Implementation details

We implemented our systems on two machines, one to emulate the law enforcement server and the other to simulate the camera operations. The server is a Ubuntu desktop machine equipped with Intel Xeon CPU E5-2620 2.10 GHz×16, 64 GB of RAM, and a 512 GB SSD. The other machine is a Ubuntu laptop with Intel Core i7-6500U CPU 2.50 GHz×4 and 8 GB of RAM (it is also equipped with a front camera). The two machines are connected via a TCP/IP4 LAN over Gigabit Ethernet. Our systems are built on top of OpenFace.[1] and OpenALPR[2]

The face recognition layer of the OpenFace implementation employs package *shape_predictor_68_face_landmarks* as face predictor and *nn4.small2.v1.t7* as the network model. The package is written in Python version 2.7 and, with the aforementioned configuration, face recognition and normalization take about 600 ms on our laptop (using the default configuration). This overhead greatly affects the performance of our system, as it is sometimes larger than the cost of the cryptographic operations.

After investigating the low-level details of OpenFace, we realized that the face recognition process comprises two important phases: (i)

[1] https://github.com/cmusatyalab/openface
[2] https://github.com/openalpr/openalpr

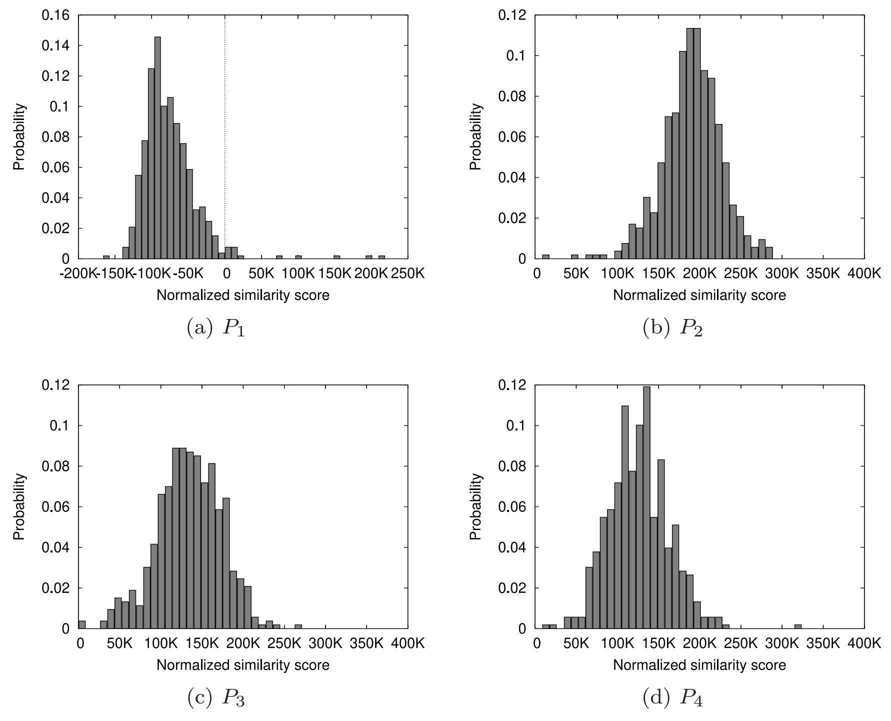


Fig. 4. Distribution of non-obfuscated similarity scores for 500 images of P_1 against one image of P_1, P_2, P_3, P_4.

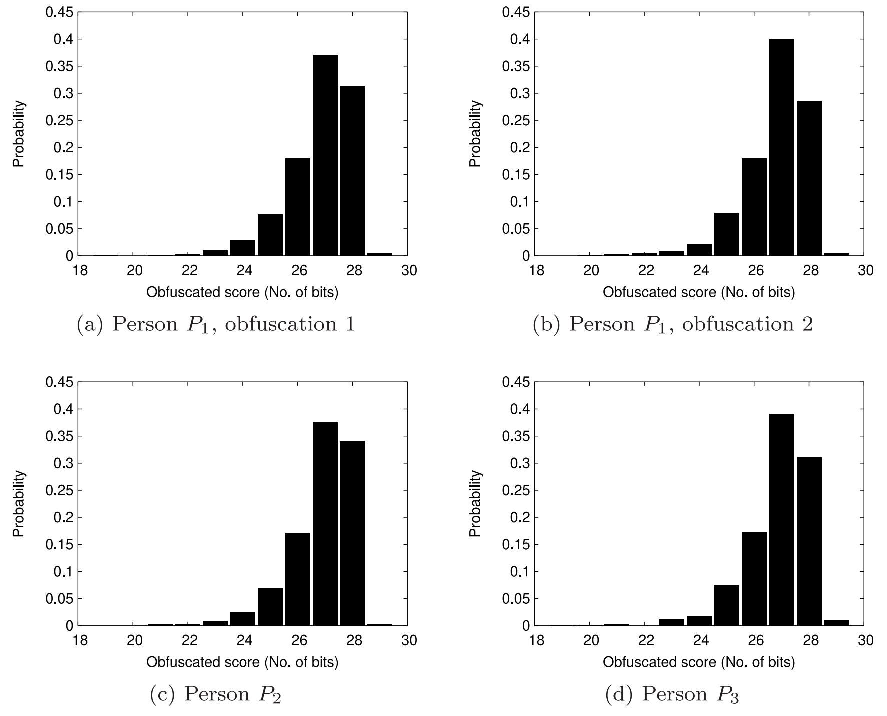


Fig. 5. Distribution of obfuscated score bit-lengths against a database of 1000 suspects ($r_1, r_2 < 2048$).

detecting and generating the bounding boxes of possible faces in a frame and (ii) generating the 128-byte vector for each bounding box. The latter operation is performed using neural networks and is easily parallelizable. On the other hand, the identification of the bounding boxes employs the `dlib` library and uses a combination of Histogram of Oriented Gradient (HOG) and Support Vector Machine (SVM) algorithms. This combination brings many benefits in terms of recognition accuracy, but it cannot be parallelized.

To mitigate this problem and leverage the power of all available CPU cores, we decided to split the recognition workload equally across the different cores. In particular, instead of trying to parallelize each step of the recognition pipeline, we assign each video frame to one of the CPU cores. Despite increasing the overall face detection time, we gain a lot in terms of processing throughput (frames per second). To get reasonable face recognition times, we have used a resolution of 640 × 480 pixels and an upsampling parameter of 1 in our implementation.

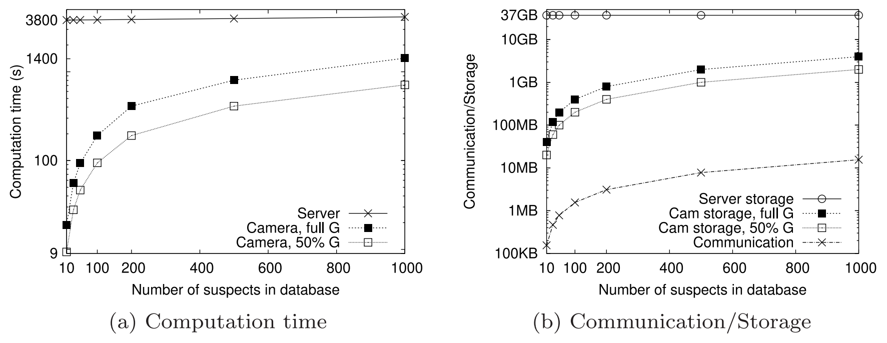


Fig. 6. Offline cost.

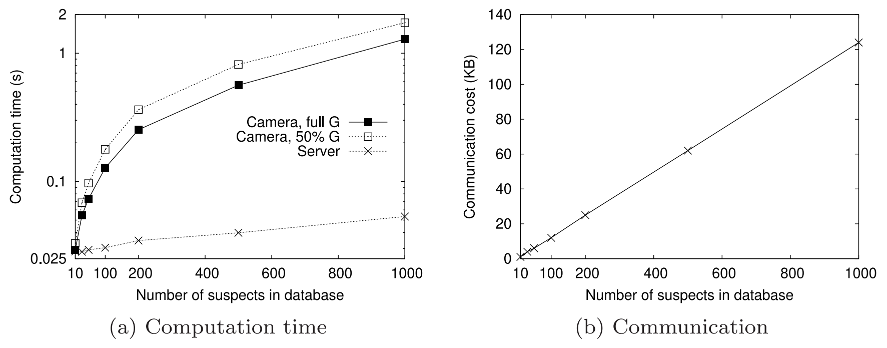


Fig. 7. Online cost.

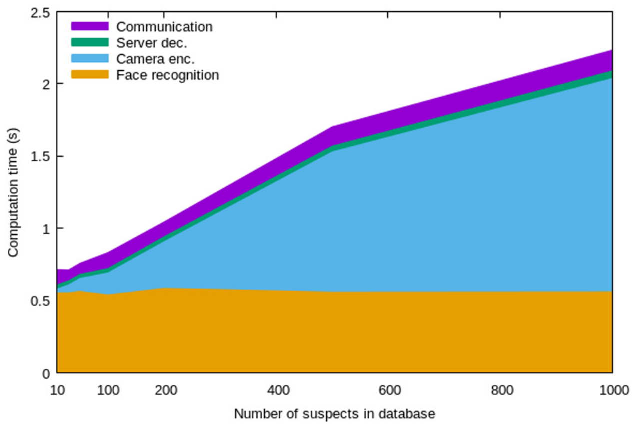


Fig. 8. Round Trip Time for a positive face match.

License plate recognition was implemented with the OpenALPR library. Similar to face recognition, the system is implemented as a client/server application on the same hardware configuration. The server's input is a *.txt* file, containing a list of the suspects' license plate numbers. We run our experiments using different database sizes, varying from $M = 10$ to 3000. On the client side, the input is typically a live video stream from the device's camera. However, the input can also include stored video sequences and still images. In our experiments, we set the maximum number of characters on a license plate to be $N = 8$. The camera (client) application sets the preferred *fault tolerance* threshold l, which can be equal to $l = 0$ (100% match) or $l = 1$ (off-by-one errors).

The cryptographic layer (elliptic curve ElGamal) for both methods was implemented in C, using the BIGNUM library of OpenSSL (version 1.1.0g). We also used SWIG to connect C with Python (version 4.0.1). We set the order of the elliptic curve to be a 256-bit prime number, as per NIST's recommendations [31]. As a result, all ciphertexts, which consist of two elliptic curve points, require 128 bytes of storage/communication. Under this C/Python environment, the average time for encryption, decryption, and scalar-point multiplication (with 256-bit scalars) is about 0.23 ms. On the other hand, point addition takes only about 0.02 ms. For each reported result, we run the experiment 4 times and plot the average time. Finally, our implementation leverages the parallel computing abilities of the two multi-core machines, since all our algorithms are easily parallelizable. The source code of our implementations is available online for both face recognition[3] and license plate recognition.[4]

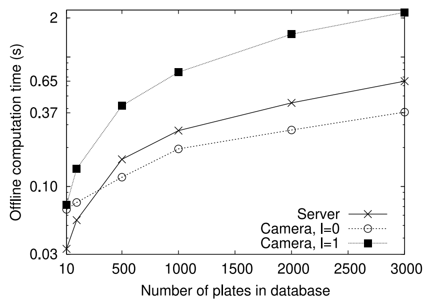
Next, we discuss the results of our experimental evaluation for the two use-cases. The reported times correspond to actual measurements collected from the two implementations on the separate devices (laptop and workstation).

9.2. Face recognition

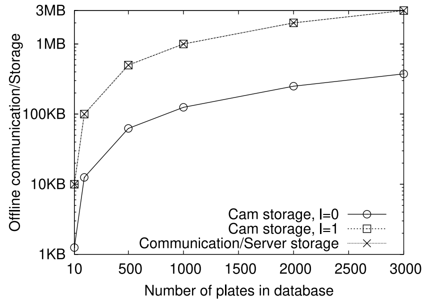
Starting with the offline phase, we first evaluate the computation and communication/storage costs at both the camera and the server, as a function of the database size M. To this end, Fig. 6 illustrates the CPU time at all parties. G represents the database of precomputed values, so the bottom curve of the plot corresponds to the cost where only 50% of the precomputations are actually performed. The cost at the camera is clearly linear in M and is dominated by the computation of the terms $y_{i,j} \cdot \mathsf{Enc}(-2x_{i,j})$, as explained in Section 6. This is, by any means, an acceptable cost, as it is incurred only once and can terminate within a few minutes.

At the server-side, the offline cost includes the generation of the suspects' feature vectors from the corresponding images (OpenFace), the normalization of their representations (our algorithm), and the generation of the encrypted database that is sent to the cameras. Nevertheless, these costs are not evident in Fig. 6, as they are dominated by

[3] https://github.com/mahdihbku/BlindGuardian
[4] https://github.com/mahdihbku/BlindCarSeeker

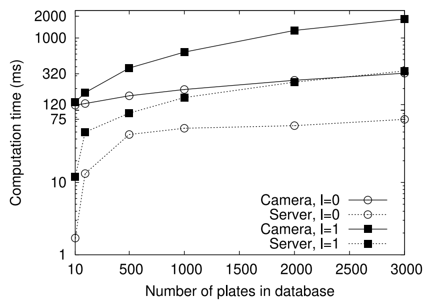


(a) Computation time

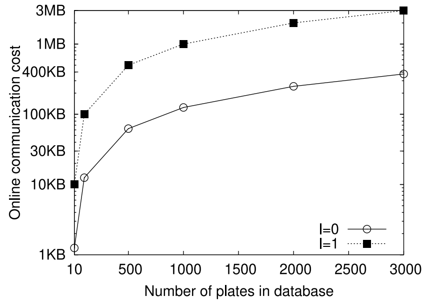


(b) Communication/Storage

Fig. 9. Offline cost.



(a) Computation time



(b) Communication

Fig. 10. Online cost.

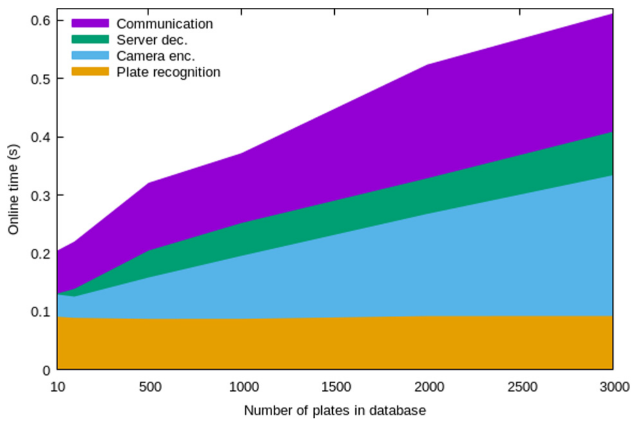


Fig. 11. Round Trip Time for a positive license plate match ($l = 0$).

the cost of the precomputations for the discrete log lookup table. This operation necessitates over an hour of compute time, but is crucial in our system because it speeds up considerably the decryption operations at the server. More importantly, this is a one-time cost that is incurred before the system becomes operational.

Fig. 6(b) depicts the offline communication/storage cost at the two devices. The compact representation of elliptic curve points makes it feasible to store the entire database G at the camera with only 4 GB of main memory. On the other hand, the cost at the server is again dominated by the discrete log lookup table, whose size is equal to 37 GB. However, this is a trivial requirement for today's state-of-the-art servers. Finally, the offline communication cost entails the transmission of the encrypted database and remains under 10 MB, even for a database of 1000 suspects.

In the next set of experiments, we evaluate the online cost of our approach, as a function of the database size M. First, Fig. 7 shows the online CPU time at all parties. Clearly, the cameras absorb most of the computational cost, since they have to compute the encrypted similarity scores for every suspect in the database. Nevertheless, the online cost is order of magnitudes lower compared to existing approaches, and remains below 1 s for databases of up to 500 suspects.

A notable observation that motivates the partial storage of G (as explained in Section 6) is that the performance penalty from storing 50% of G is not significant. In particular, for $M = 100$, the CPU time at the camera when the full G is available is 155 ms, and it only increases by 35% (to 210 ms) when 50% is available. Finally, a very promising result of our implementation is the online computation cost at the database server. For $M = 100$ the cost is just 34 ms, while for $M = 1000$ it only raises to 50 ms. As mentioned previously, the database server is the bottleneck in a wide-scale video surveillance system, because it may potentially process thousands of captured faces every second.

Fig. 7(b) illustrates the online communication cost for our system. It involves a single round of communication, where the camera transmits M encrypted similarity scores to the server. For a database of 1000 suspects, this entails a communication cost of just 128 KB.

Our previous experiments focused only on the cryptographic overhead of the privacy-preserving face recognition system. Alternatively, Fig. 8 illustrates the true Round Trip Time (RTT) for detecting a suspect. It includes face recognition and detection at the camera, all the cryptographic operations at both the camera and the server, and the required communication that includes sending the suspect's image from the camera to the server. For a database of 100 suspects, the RTT is less than 0.8 seconds (with a precomputation of the entire database G), while for $M = 1000$ the RTT is approximately 2.2 seconds. Nevertheless, a fixed portion of the RTT (around 0.6 s) is consumed on non-cryptographic operations, namely the face recognition and detection by the OpenFace software. A better combination of hardware/software at the surveillance cameras could improve that cost considerably.

9.3. License plate recognition

Fig. 9(a) depicts the offline computation cost for the server and client applications. As explained in Section 7, the server's sole offline

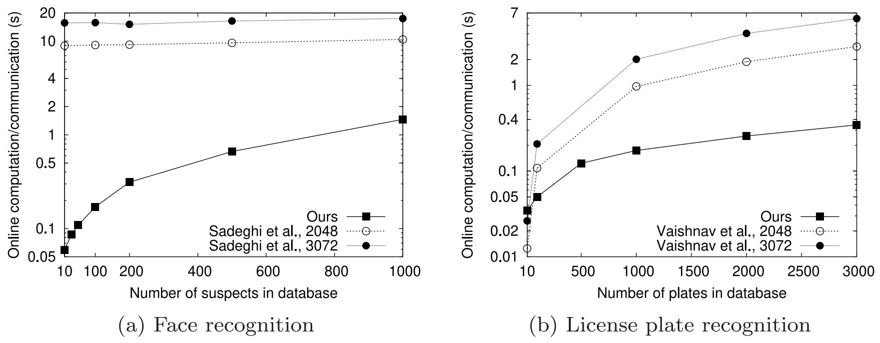


Fig. 12. Online cost vs. state-of-the-art protocols.

task includes the generation of the encrypted database, which comprises $N \times M$ encryption operations. On the other hand, the camera has to generate the extended database (depending on the fault tolerance value l), and precompute $|\mathcal{A}| \cdot N$ ciphertexts that are utilized during the online phase. In terms of communication, Fig. 9(b) illustrates that cost is very low, even for a database of 3000 entries. The cost is dominated by the transmission of the encrypted database from the server to the camera. Similarly, the storage space required at both the camera and the server is negligible, and remains below 3 MB in all cases.

Next, we evaluate the online cost of our approach. Fig. 10(a) shows the computation time required by the server and the client to compare one captured license plate against the entire suspects' database. As expected, this cost increases considerably when we incorporate the fault tolerance mechanism. Without fault tolerance ($l = 0$), the computation time on the laptop (client) is 325 ms for $M = 3000$. This cost includes frame processing and license plate extraction (92 ms), and the computation/obfuscation of the similarity scores (232 ms). The server's CPU cost involves the partial decryption of M ciphertexts, which takes only 75 ms. Note that both the client and server applications are multi-threaded and utilize all available cores. The online communication cost (Fig. 10(b)) is simply the cost of transmitting the encrypted similarity scores to the server. Without fault tolerance, the number of ciphertexts is equal to M, which requires 375 KB of online communication. On the other hand, when $l = 1$, the cost is increased by a factor of $N = 8$.

Fig. 11 illustrates the Round Trip Time for detecting a suspect license place. It includes all the performed operations, such as OpenALPR's license plate recognition, the cryptographic computations at the camera and server, and the communication time for transmitting the encrypted similarity scores and the image of the captured license plate. When the suspects' database is large, the RTT is dominated by the cryptographic operations at the camera and the communication cost. Nevertheless, even for a database of 3000 license plates, the RTT is just 612 ms.

9.4. Comparison with the state-of-the-art

Finally, in Fig. 12 we compare our methods with the current state-of-the-art solutions, namely Sadeghi et al. [3] for face recognition and Vaishnav et al. [5] for license plate recognition. The graphs represent the combined online computation and communication cost that is required to match a single suspect against the server's database (assuming a 10 Mbps bandwidth between the cameras and the server). Both our competitors are implemented on top of Paillier's homomorphic cryptosystem. To this end, we run two sets of experiments under different security levels, i.e., with a 2048-bit and 3072-bit RSA modulus. According to NIST's recommendations [31], for 128 bits security, RSA-based algorithms necessitate a 3072-bit modulus (and a 2048-bit modulus for 112 bits security). On the other hand, our approach offers 128 bits security with the selected 256-bit curve. Note that, for a fair comparison, both protocols were implemented using multi-threading, in order to take advantage of the multi-core CPU machines.

Sadeghi et al. improved the online time by leveraging precomputations, wherever possible, and packing multiple distance scores in a single ciphertext. This reduces significantly the number of expensive public key operations. Furthermore, Vaishnav et al. is based on an improved Paillier implementation proposed by Jost et al. [32]. Their algorithms optimize the encryption operation by employing precomputations, whereas the decryption is improved by reducing the size of the exponent to 320 bits, instead of 2048 or 3072. For Vaishnav et al.'s method, we run the experiments without the trusted third party, in order to match our proposed architecture.

Fig. 12(a) shows that the required time to match a single face against a database of 500 suspects is around 9.5 s for 112 bits security, and more than 17 s for 128 bits security. The cost is dominated by (i) the projection phase at the server (computations) and (ii) the communication cost that involves multiple rounds of communication and more than 5.8 MB of data exchange. On the other hand, our method reports only 663 ms for 500 suspects. As such, for the same security level, our protocol reduces the overall cost by a factor of 26.

Fig. 12(b) shows that, for a database of 3000 license plates, Vaishnav et al. requires 2.84 s and 6.1 s for 112 and 128 bits security, respectively. The cost is dominated by the decryption operation at the server and the communication overhead. Our method necessitates only 346 ms of online cost, i.e., a factor of 18 improvement.

10. Conclusion

In this paper, we introduced the first near real-time privacy-preserving video surveillance system. We started by designing a general framework for privacy-preserving surveillance that has several advantages over the existing state-of-the-art approaches. The benefits of our framework stem mainly from two design decisions. First, the encrypted suspects' database is distributed to all surveillance cameras, which facilitates the use of extensive precomputations that reduce significantly the computation of the encrypted similarity scores. Second, the protocol involves a single round of communication, where the server receives a random permutation of obfuscated similarity scores that instantly reveal the suspect identification result. We applied our framework to two distinct use-cases, namely, face recognition and license plate recognition, and implemented the corresponding system prototypes. We performed an extensive experimental evaluation of the two systems and our results show that, compared to the current state-of-the-art approaches, our protocols reduce the overall cost by a large factor.

CRediT authorship contribution statement

Elmahdi Bentafat: Conceptualization, Methodology, Software, Writing – original draft. **M. Mazhar Rathore:** Conceptualization, Methodology, Writing – review & editing. **Spiridon Bakiras:** Conceptualization, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Open Access funding provided by the Qatar National Library.

References

[1] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, T. Toft, Privacy-preserving face recognition, in: Proc. International Symposium on Privacy Enhancing Technologies (PETS), 2009, pp. 235–253.
[2] M. Osadchy, B. Pinkas, A. Jarrous, B. Moskovich, Scifi – A system for secure face identification, in: Proc. IEEE Symposium on Security and Privacy (SP), 2010, pp. 239–254.
[3] A.-R. Sadeghi, T. Schneider, I. Wehrenberg, Efficient privacy-preserving face recognition, in: Proc. International Conference on Information Security and Cryptology, 2009, pp. 229–244.
[4] A.B. Sunil, Z. Erkin, T. Veugen, Secure matching of dutch car license plates, in: 2016 24th European Signal Processing Conference (EUSIPCO), IEEE, 2016, pp. 2116–2120.
[5] H. Vaishnav, S. Sharma, A. Mathuria, Efficient implementation of private license plate matching protocols, in: International Conference on Security, Privacy, and Applied Cryptography Engineering, Springer, 2017, pp. 281–294.
[6] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inform. Theory 31 (4) (1985) 469–472.
[7] E. Bentafat, M.M. Rathore, S. Bakiras, A practical system for privacy-preserving video surveillance, in: International Conference on Applied Cryptography and Network Security, Springer, 2020, pp. 21–39.
[8] B. Amos, B. Ludwiczuk, M. Satyanarayanan, Openface: A general-purpose face recognition library with mobile applications, CMU School Comput. Sci. 6 (2016).
[9] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815–823.
[10] Openalpr: Automatic license plate recognition, 2014, URL https://github.com/openalpr/openalpr.
[11] A.C.-C. Yao, How to generate and exchange secrets, in: Proc. Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167.
[12] C. Xiang, C. Tang, Y. Cai, Q. Xu, Privacy-preserving face recognition with outsourced computation, Soft Comput. 20 (9) (2016) 3735–3744.
[13] M. Naor, B. Pinkas, Computationally secure oblivious transfer, J. Cryptol. 18 (1) (2005) 1–35.
[14] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, M. Zohner, GSHADE: faster privacy-preserving distance computation and biometric identification, in: Proc. ACM Workshop on Information Hiding and Multimedia Security, 2014, pp. 187–198.
[15] D. Evans, Y. Huang, J. Katz, L. Malka, Efficient privacy-preserving biometric identification, in: Proceedings of the 17th Conference Network and Distributed System Security Symposium, NDSS, Vol. 68, 2011, pp. 90–98.
[16] P. Gasti, J. Šeděnka, Q. Yang, G. Zhou, K.S. Balagani, Secure, fast, and energy-efficient outsourced authentication for smartphones, IEEE Trans. Inf. Forensics Secur. 11 (11) (2016) 2556–2571.
[17] C. Karabat, M.S. Kiraz, H. Erdogan, E. Savas, THRIVE: threshold homomorphic encryption based secure and privacy preserving biometric verification system, EURASIP J. Adv. Signal Process. 2015 (1) (2015) 71.
[18] K. Zhou, J. Ren, PassBio: Privacy-preserving user-centric biometric authentication, IEEE Trans. Inf. Forensics Secur. 13 (12) (2018) 3050–3063.
[19] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proc. ACM Symposium on Theory of Computing (STOC), 2009, pp. 169–178.
[20] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Proc. International Conference on the Theory and Applications of Cryptographic Techniques, 1999, pp. 223–238.
[21] H. Vaishnav, A. Mathuria, Fast private license plate matching using symmetric homomorphic encryption, in: 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), IEEE, 2018, pp. 1–6.
[22] J. Dyer, M. Dyer, J. Xu, Practical homomorphic encryption over the integers for secure computation in the cloud, Int. J. Inf. Secur. 18 (5) (2019) 549–579.
[23] J.T. Trostle, A. Parrish, Efficient computationally private information retrieval from anonymity or trapdoor groups, in: Proc. International Conference on Information Security (ISC), 2010, pp. 114–128.
[24] H.W. Lenstra, A.K. Lenstra, L. Lovfiasz, Factoring polynomials with rational coefficients, Math. Ann. 261 (1982) 515–534.
[25] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: Closing the gap to human-level performance in face verification, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1701–1708.
[26] G. Bradski, A. Kaehler, Opencv, Dr. Dobb's J. Softw. Tools 3 (2000).
[27] R. Smith, Z. Podobny, et al., Tesseract OCR, 2005, URL https://github.com/tesseract-ocr/tesseract.
[28] A. Acar, H. Aksu, A.S. Uluagac, M. Conti, A survey on homomorphic encryption schemes: Theory and implementation, ACM Comput. Surv. 51 (4) (2018) 79.
[29] C. Schnorr, Efficient signature generation by smart cards, J. Cryptol. 4 (3) (1991) 161–174.
[30] Y. Lindell, B. Pinkas, Secure multiparty computation for privacy-preserving data mining, J. Priv. Confid. 1 (1) (2009).
[31] E. Barker, NIST special publication 800-57, in: NIST Special Publication, Recommendation for Key Management–Part 1: General (Revision 5), Vol. 800, (57) 2020, pp. 1–171.
[32] C. Jost, H. Lam, A. Maximov, B.J. Smeets, Encryption performance improvements of the paillier cryptosystem, IACR Cryptol. EPrint Arch. 2015 (2015) 864.