

# **SCiFI – A system for Secure Computation of Face Identification**

Margarita Osadchy, Benny Pinkas, Ayman  
Jarrous, Boaz Moskovitch  
University of Haifa

# Face recognition technology



face identification  
(surveillance)  
arbitrary conditions



face identification  
(login)  
controlled conditions

# We focus on the surveillance problem



surveillance

Example scenario:

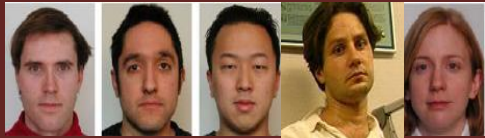
- a government has a list of suspects
- wants to identify them in a crowd

# Face recognition in surveillance



- **Privacy problem:** the ubiquity of surveillance is a major concern for the public
  - Can be misused to track people regardless of suspicion
  - Can be combined with a universal database linking faces to identities (e.g., drivers' license photos)

# A solution to the privacy concern



match / no match



Operator

Store the suspects database at the client

**Not acceptable** if the list of suspects is confidential, as is often the case.



# Our approach: protecting the privacy of the public and the confidentiality of the data



Secure computation



match / no match



only learn match / no match

# System architecture

## Client

Acquires an image

Generates representation of image

Runs secure protocol

Output: match / no-match

## Server

Input: set of images of suspects

Runs secure protocol

Output: match / no-match



# System architecture

## Client

Acquires an image

Generates representation of image

Runs secure protocol

Output: match / no-match

## Server

Input: set of images of suspects

Runs secure protocol

Output: match / no-match



Protocol enforces an upper bound on the size of the database used by the server.



# The Problem

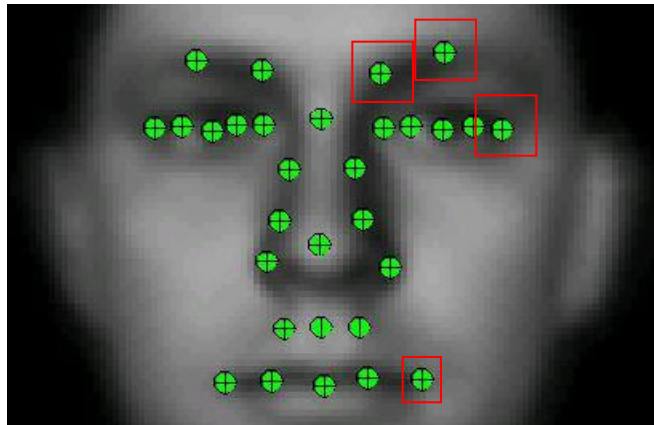
- Exact / fuzzy match
  - Secure computation of *exact* matches is well known.
  - Face identification is *fuzzy*. A match is between *close*, but *not identical*, images.
- Continuous / discrete math
  - Face recognition algorithms use *continuous* face representations, and complex measures of similarity.
  - Secure computation is always applied to *discrete* numbers. Best with linear operations.
  - Simple quantization of face recognition algorithms results in poor performance.

# Our Contributions

- A new and unique **face identification algorithm**
  - **Specifically designed** for secure computation
  - Has state-of-the-art **recognition** performance
  - Assumes only a **single** image is known per suspect
- A **secure protocol** for computing face identification
- **SCiFI** - A system implementing the protocol
- Previous work [EFGKLT09]: secure computation of the well known Eigenfaces face recognition algorithm.
  - Performance of eigenfaces is inferior to state-of-the-art.
  - The secure protocol is less efficient than ours.

# New Face Representation: Patch-Based Face Representation

- A face is represented by a collection of informative patches:



⊕ Patch centers

□ Patch size –  
could vary

- Assume that the face is represented by  $p$  patches.

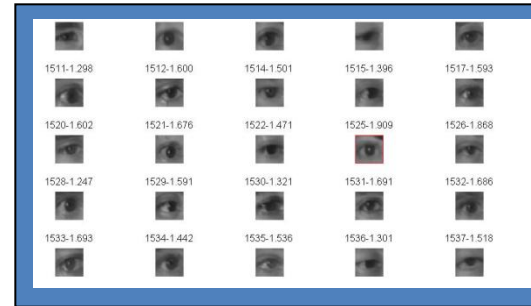
# Gallery / Dictionary



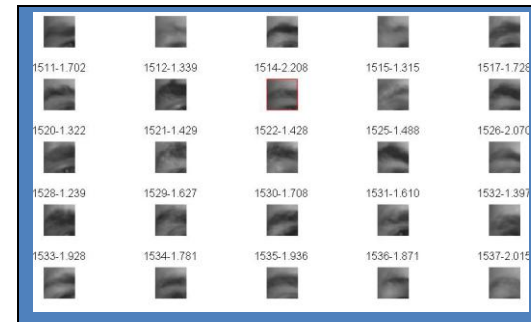
A **public** database (**gallery**) of  $N$  faces

⇒ A dictionary of  $N$  values for each patch

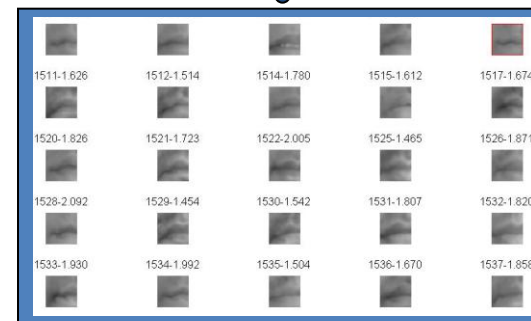
1



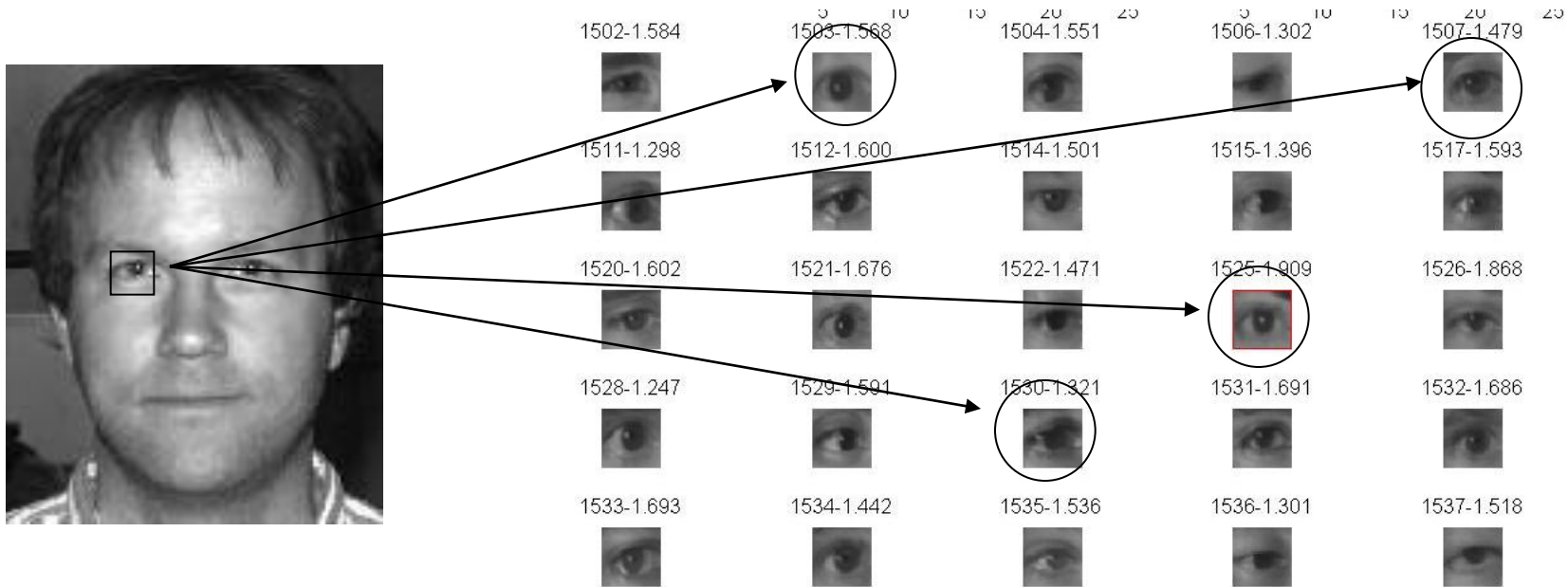
2



p

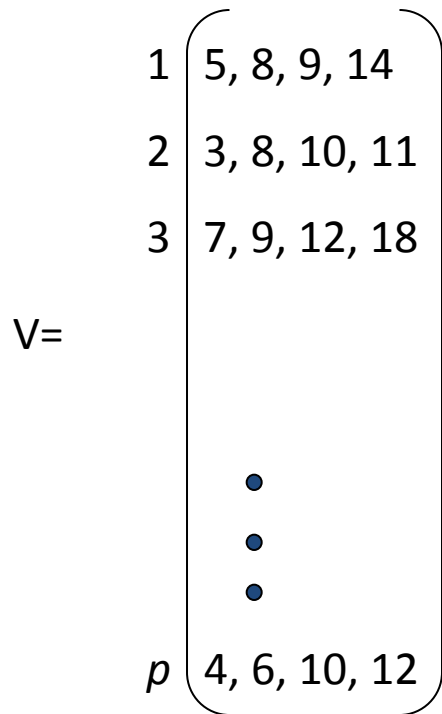
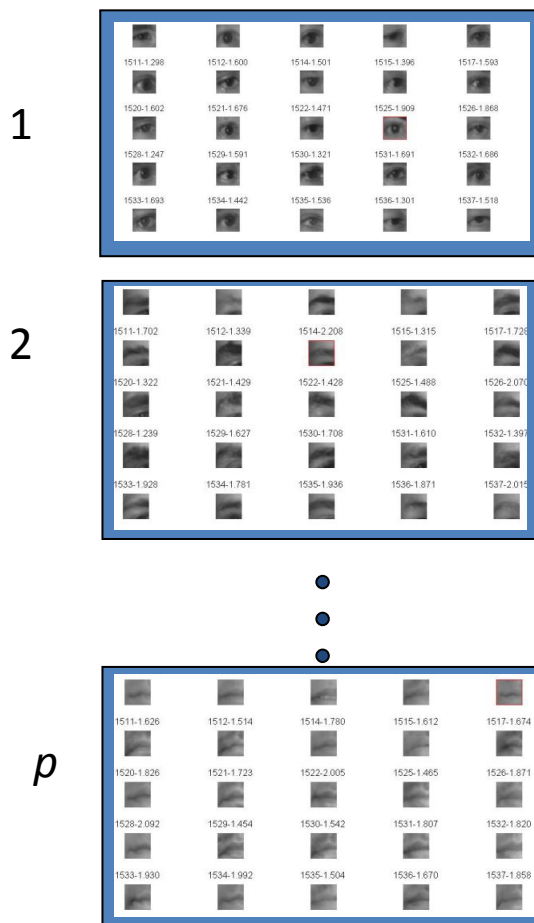
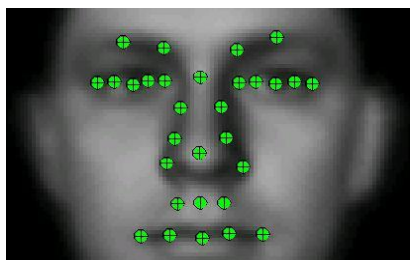


# Indexing



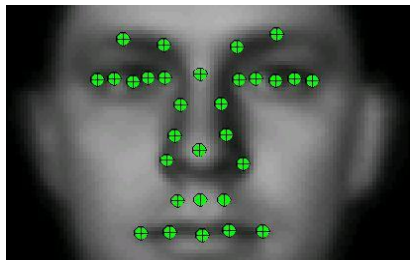
Each patch is represented by the **4 closest patches** in the dictionary.

# Representing a face



For each of the  $p$  patches, store indices of the 4 closest patches in the dictionary.

# Representing a face



$$V = \begin{pmatrix} 1 & 5, 8, 9, 14 \\ 2 & 3, 8, 10, 11 \\ 3 & 7, 9, 12, 18 \\ \vdots & \vdots \\ p & 4, 6, 10, 12 \end{pmatrix}$$

For each of the  $p$  patches, store indices of the 4 closest patches in the dictionary.

**Representation:** vector with  $p$  entries, each with 4 values in the range of  $[1, N]$ .

Alternatively, a **binary representation:** a binary vector of  $p \cdot N$  bits, where  $4p$  of the bits equal 1.



# Similarity between faces

- We define the difference between faces as the **set difference** between their representations  
$$\Delta(A,B) = |A \cup B| - |A \cap B|$$
- Set difference  $\equiv$  **Hamming distance between binary representation of faces**
- Secure computation of Hamming distance is easy [JP09]

# Cryptographic Protocol

- **Functionality:**
  - Client and server each have a binary vector representing a face.
  - Output 1 iff Hamming distance  $<$  threshold.
- **Tools**
  - Additively homomorphic encryption
    - Given  $E(x)$ ,  $E(y)$  can compute  $E(x+y)$
  - Oblivious transfer
    - A two-party protocol where receiver can privately obtain one of two inputs of a sender

# The protocol in a nutshell

(details and proof in the paper)

- Inputs are vectors  $w = w_0, \dots, w_{m-1}$ ;  $w' = w'_0, \dots, w'_{m-1}$ .
- Client sends  $E(w_0), \dots, E(w_{m-1})$
- Server uses homomorphic properties
  - To compute  $E(w_0 \oplus w'_0), \dots, E(w_{m-1} \oplus w'_{m-1})$
  - To sum these values and obtain  $E(d_H(w, w')) = E(d)$
- Server chooses random  $R$ ; sends  $E(d+R)$  to client
- Client decrypts  $E(d+R)$ , reduces the result mod  $m+1$ .
- Both parties run a  $1\text{-out-of-}(m+1)$  OT, where client learns 1 if Hamming distance  $<$  threshold.

# Optimizations

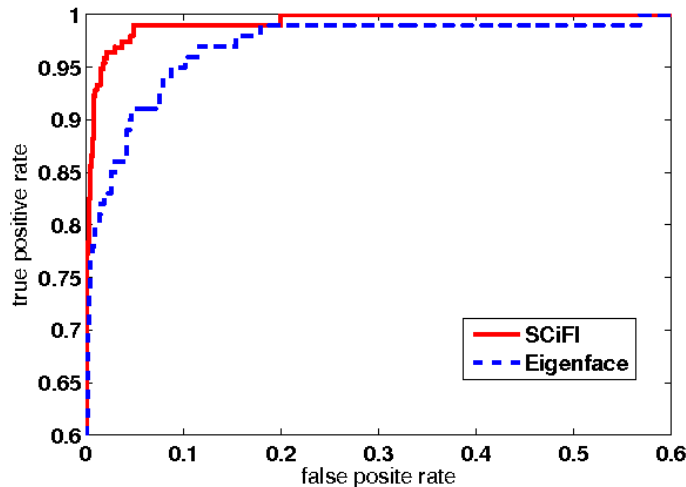
- **Main goal:** minimize *online* latency, to identify suspects in real time.
- **Methods used:**
  - Change protocol s.t. oblivious transfer and most communication can be done **before** image is recorded.
  - Prefer more efficient homomorphic operations  
*addition*  $\ll$  *encryption*  $<$  *subtraction*

# Online overhead

- A face is represented by a 900 bit vector.
- **Overhead after the client captures an image:**
  - Client sends 900 bits to server
  - For every image in server's database
    - Server performs 450 homomorphic additions
    - Server sends a single encryption to client
    - Client decrypts the encrypted value
    - Run a *preprocessed* OT: client sends 8 bits to server; server sends 180 bits to client.

# Recognition experiments

- Ran experiments with *standard databases* used by the face recognition community.
- Tested **robustness** to illumination changes, small changes in pose, and partial occlusions.



Robustness compared to Eigenfaces



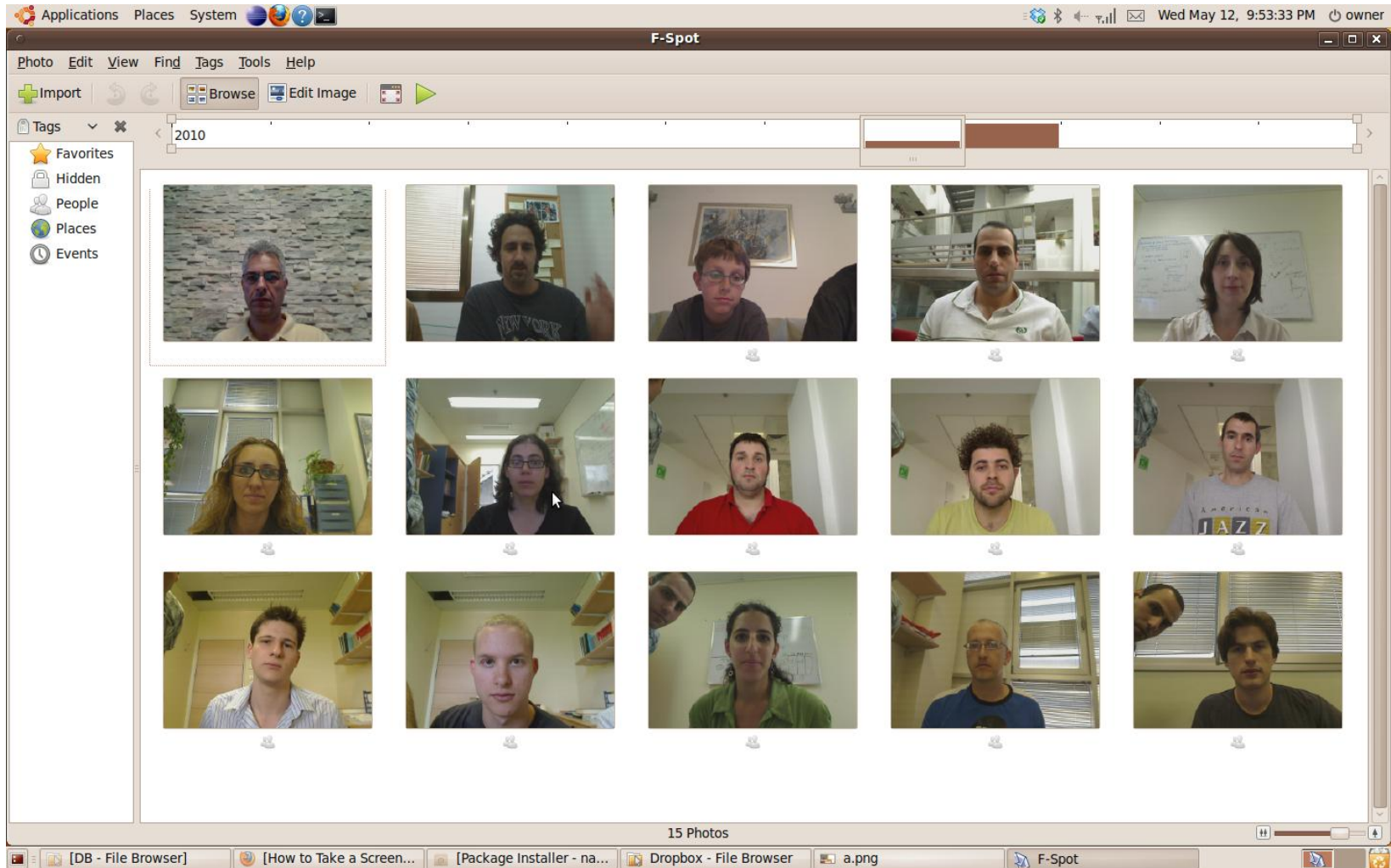
Robustness to partial occlusions

# Implementation

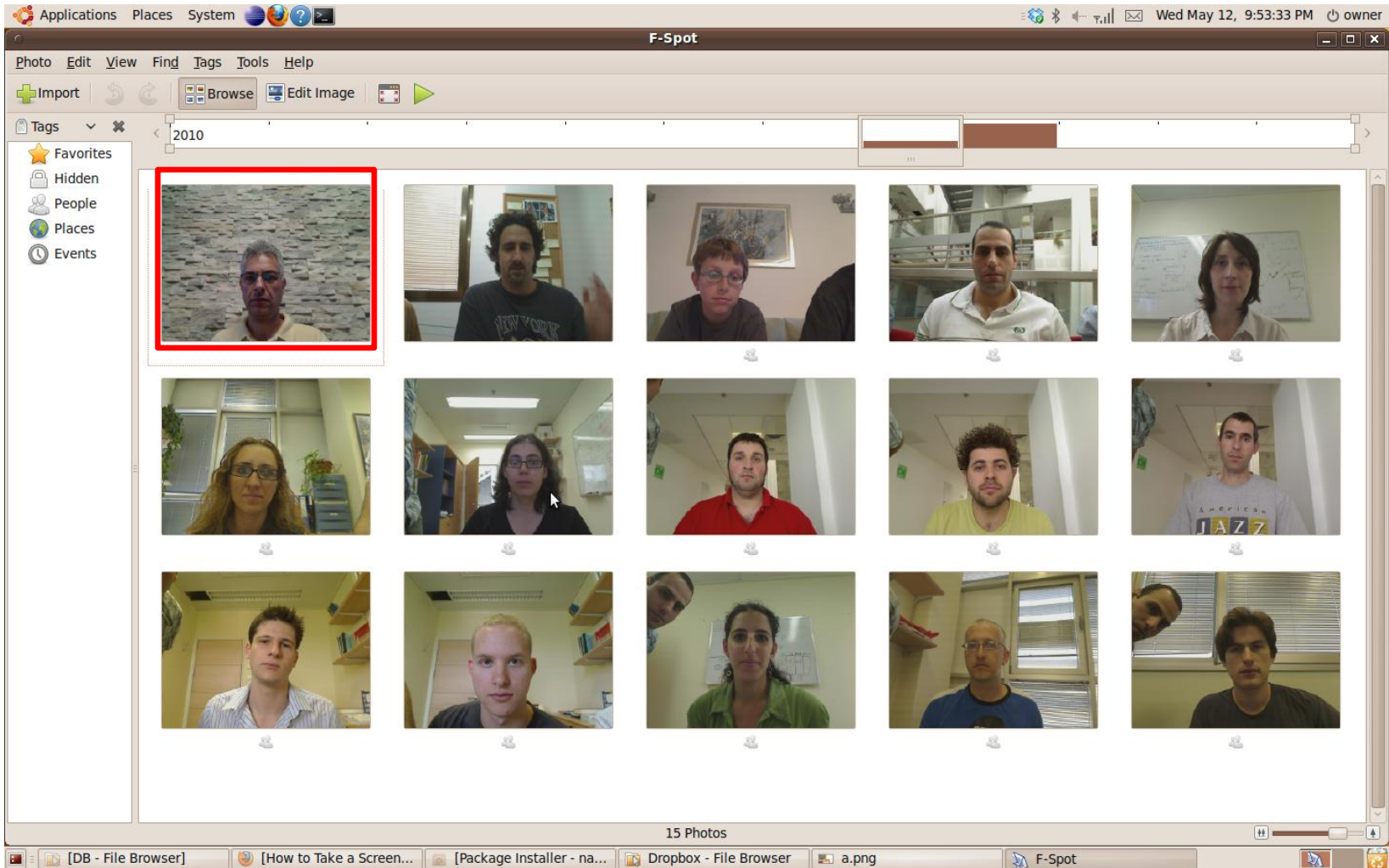
- **Face recognition** part (generating representations of images)
  - Implemented in **Matlab**, ran using Matlab Java builder.
- **Cryptographic** protocol
  - Implemented in **Java**, using Paillier and ElGamal based OT.
- Timing on Linux servers:
  - ~0.3 sec to compare to a single image in the database
  - An Implementation in C will be much faster
  - Easily parallelizable



# The database



# The suspect



# The suspect





# An image is obtained by the client

The screenshot displays a Linux desktop environment with several open windows:

- Client Viewer:** Shows a photograph of a man with grey hair and a mustache, wearing a brown shirt. Below the image are three buttons: "Capture", "Check", and "Proceed".
- Terminal:** A window with a menu bar (File, Edit, View, Terminal, Help) and a large empty text area. Three red arrows point from the text "no glasses", "slightly different pose", and "different clothes" to the man's face in the Client Viewer window.
- Server:** A window titled "Server" with a "Status:" section containing the following text:

```
Reading database
Start connection and waiting for request
Start Preprocessing
Read client encrypted random input
Initialize OTs
Start Execution
```
- System Monitor:** A window titled "System Monitor" with a menu bar (Monitor, Edit, View, Help) and tabs for System, Processes, Resources, and File Systems. It displays a "CPU History" graph showing CPU usage over time.

The desktop background is a solid orange color. The system tray at the bottom shows icons for [DB - File Browser], [How to Take a Scre...], System Monitor, Terminal, Server, Terminal, Client Viewer, and a system status icon. The top panel shows "Applications", "Places", "System", and a clock displaying "Wed May 12 9:01:26 PM" with the user name "owner".

# Facial features are recognized

The screenshot displays a Linux desktop environment with several open windows. The top panel shows the system menu with 'Applications', 'Places', and 'System' options. The system tray on the right indicates the date and time as 'Wed May 12, 9:01:42 PM' and the user as 'owner'. A red circle highlights the time '9:01:42 PM'.

The main window is 'Client Viewer', which contains a video feed of a man's face. Below the video are three buttons: 'Capture', 'Check', and 'Proceed'. A smaller window titled 'Figure 1' is overlaid on the video, showing a close-up of the man's face with several colored dots (red, green, blue, purple) marking facial features for recognition. This window has 'OK' and 'Cancel' buttons.

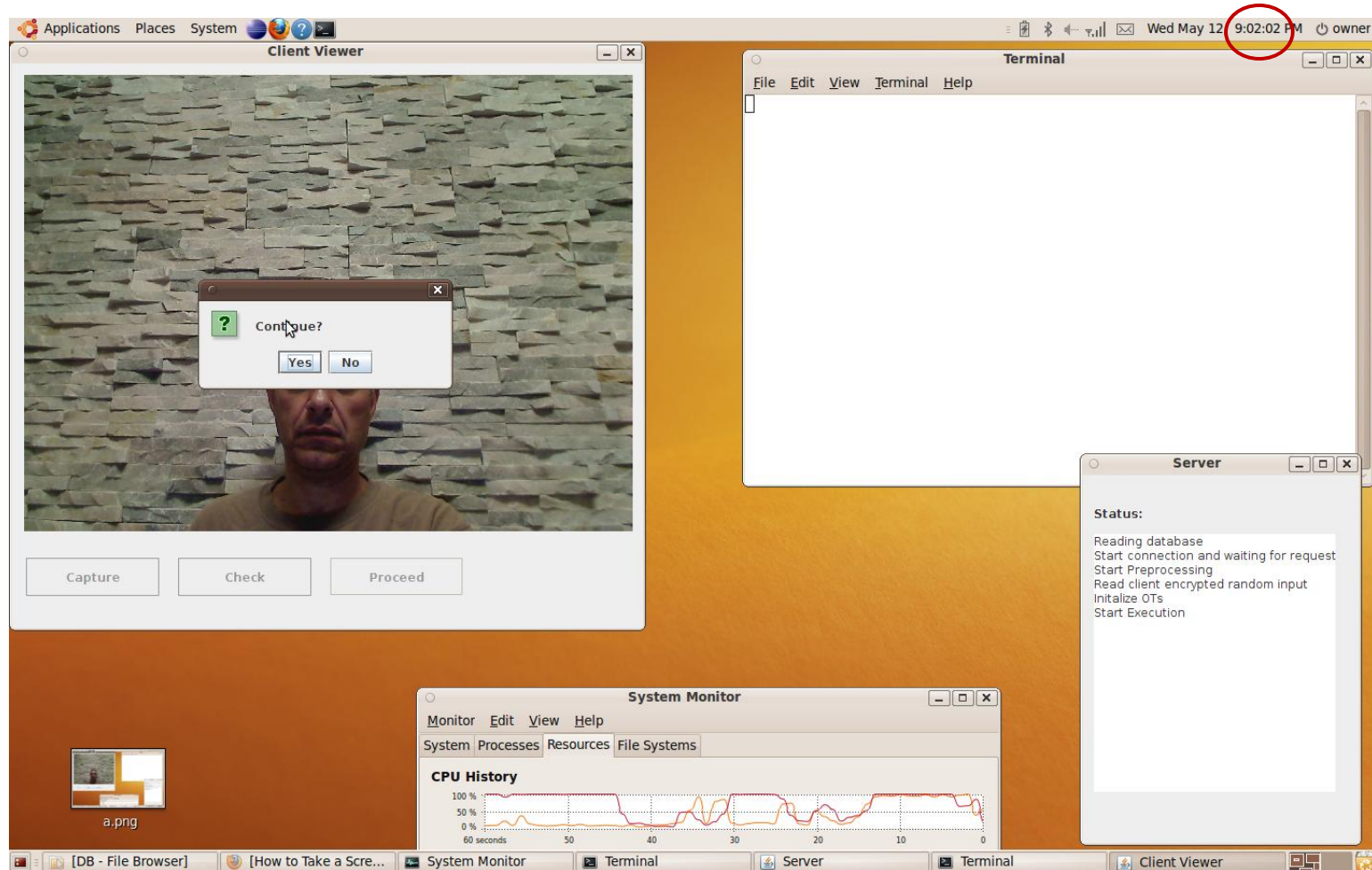
To the right, a 'Terminal' window is open, displaying a list of status messages:

```
Status:  
Reading database  
Start connection and waiting for request  
Start Preprocessing  
Read client encrypted random input  
Initialize OTs  
Start Execution
```

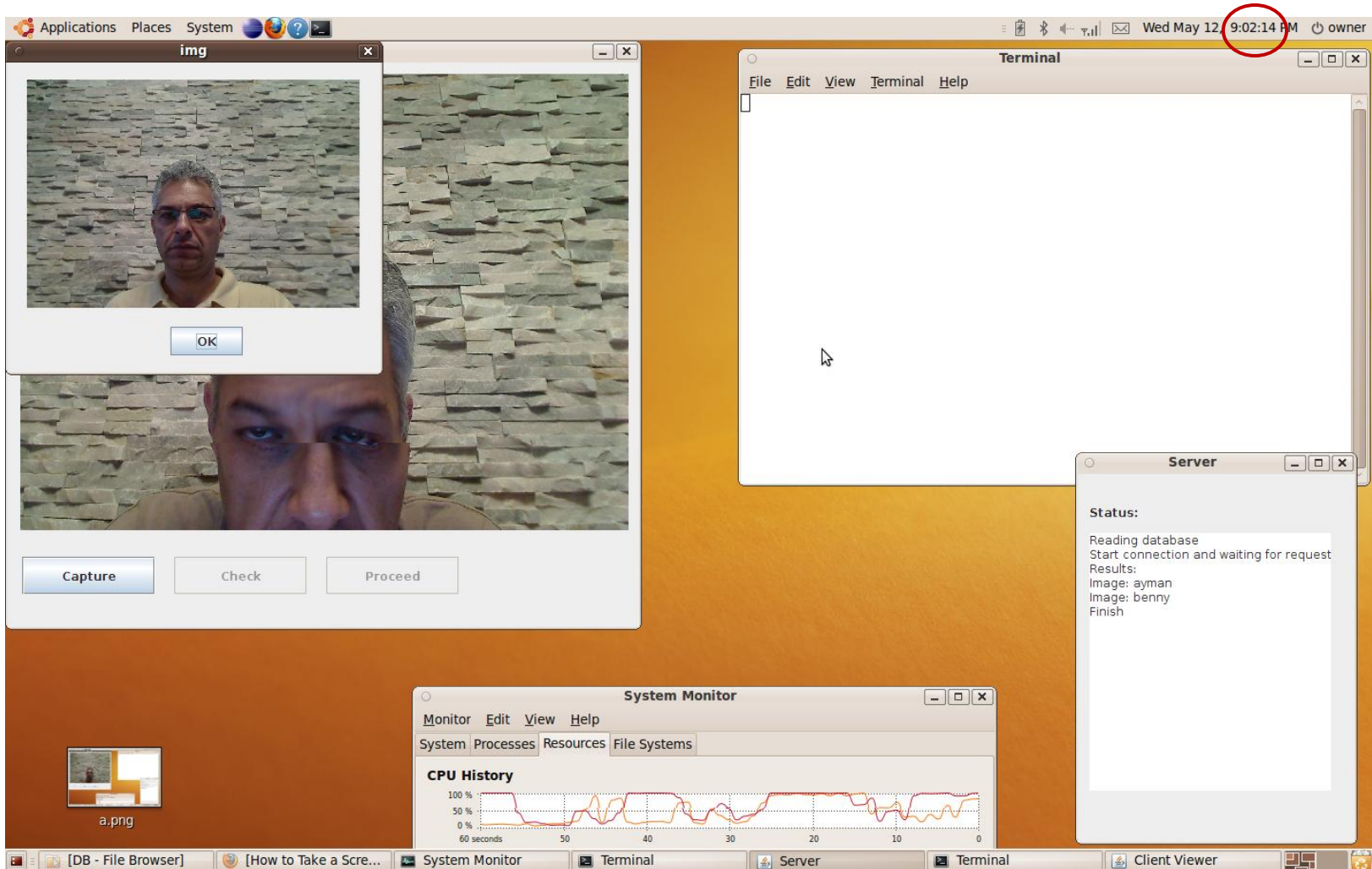
At the bottom, a 'System Monitor' window is open, showing a 'CPU History' graph with a red line fluctuating between 0% and 100% over a 60-second period. The desktop background is orange, and a file named 'a.png' is visible on the desktop.

The taskbar at the bottom shows the following windows: '[DB - File Browser]', '[How to Take a S...', 'System Monitor', 'Terminal', 'Server', 'Terminal', 'Client Viewer', and 'Figure 1'.

# Face representation is ready



# Secure protocol is run, a match is found





Live demo available upon  
request

# Conclusions

- **Goal:** Face recognition based surveillance, respecting subjects privacy.
- **Means:**
  - A new and unique face identification algorithm
    - State of the art robustness
    - Suitable for secure computation
  - A secure protocol with optimized online runtime
  - Experiments verifying robustness and performance