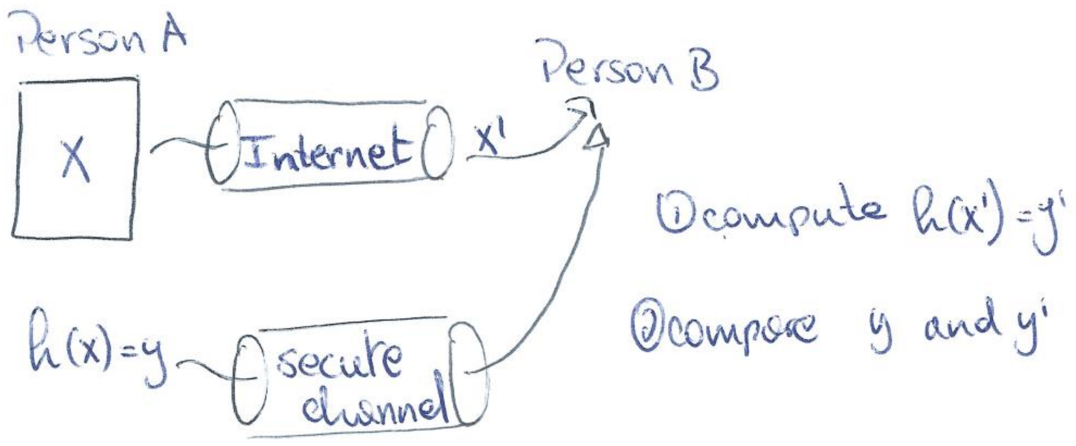
Multimedia security and privacy

Section 2

(2.1) Explain the main differences between content fingerprinting (robust perceptual hashing) and cryptographic hashing

- A cryptographic function has the following properties:
  - Noted: h(X) = y
  1. Input (X) is a binary string of any length (padding may be required)
  2. Output (y) is a binary string of a given length
  3. Changing a single bit in X, will change at least 50% of the bits of y
  4. From y, it is impossible to find the corresponding X
  5. It is computationally impossible to find collisions
  6. Some hashing functions may require the use of keys

Usage: Integrity
(Authentication if key is used)

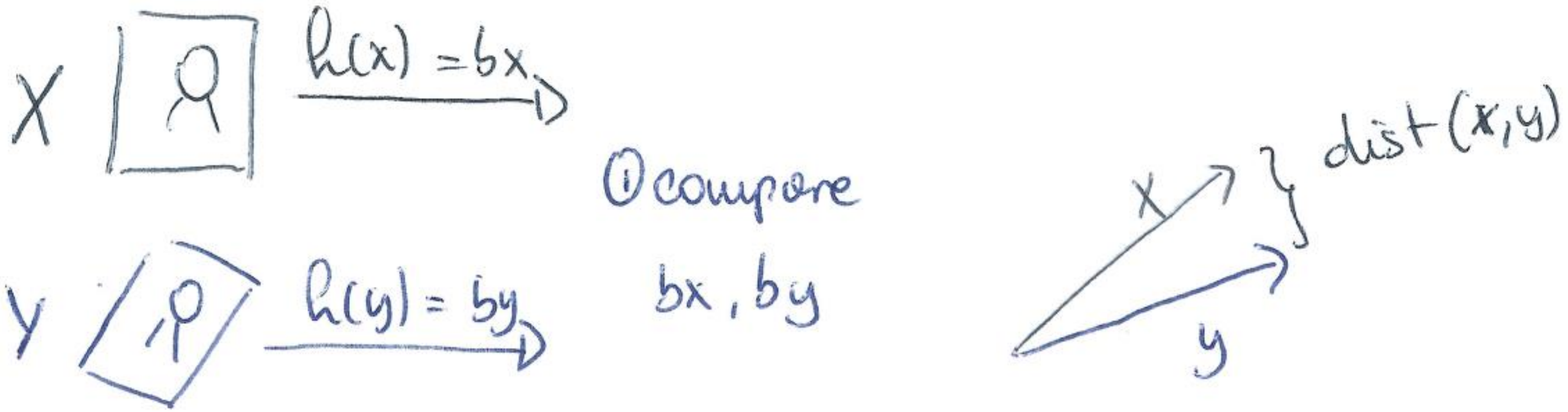


(2.1)

- A content fingerprinting function has the following properties:
  - Noted: h(X) = bx

1. Input (X) is a binary string of any length (N)
2. Output (bx) is a binary string of a given length (L) => Short
3. Changing a single bit in X, will have small repercussions on y => Invariant
4. From y, it is impossible to find the corresponding X => Secure
5. Easy to find collisions
6. Some hashing functions may require the use of keys

Usage: Authentication and Identification



(2.1)

$$\left[\begin{matrix} | \\ \underline{bx} \\ | \end{matrix}\right]^{1}_{L} = \left[\begin{matrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_L \end{matrix}\right]^{1}_{L} = \left[\begin{matrix} w_1^t x \\ w_2^t x \\ \vdots \\ w_L^t x \end{matrix}\right]^{1}_{L} = \left[\begin{matrix} - & w_1^t & - \\ - & w_2^t & - \\ & \vdots & \\ - & w_L^t & - \end{matrix}\right]^{N}_{L} \left[\begin{matrix} | \\ \underline{X} \\ | \end{matrix}\right]^{1}_{N}$$

- Levels of security

  N >> L, since the hash is much smaller than the input data, it is impossible to recover X from bx. This also results in easy comparison between hashes.

  Entropy of input: H(X) = N*H(Xi) = N*log2(256) = N*8

  Entropy of output: H(bx) = L*H(bxi) = L*log2(2) = L*1 = L

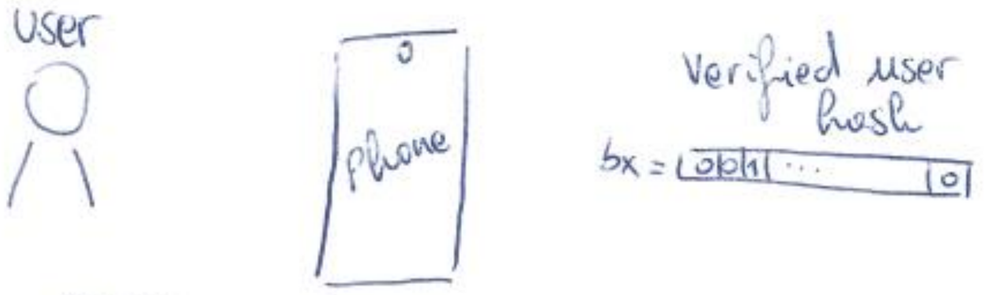
(2.2) Explain the main usage of content fingerprinting in various applications. What are the main advantages?

- Main Usages: authentication and identification

- Authentication:

  Example: a phone that is able to unlock via face recognition, must take a snap picture of a user (that wants to unlock the phone). The phone must decide “is it a verified user, or not?”

=> yes or no question



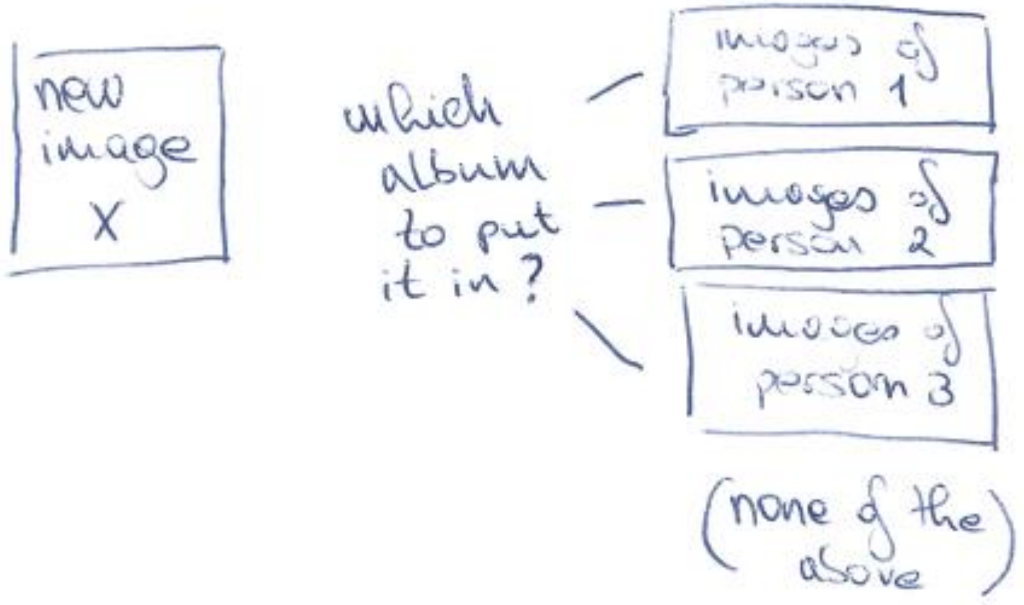
① take picture $\rightarrow y$
② compute $h(y) = b_y$
③ Compare b_x and b_y

(2.2)

- Identification:

  Example: a phone keeps albums of images of each different person in its images. When taking a new picture, it needs to identify into which album the image must go.

  => Must decide which album, or none at all



(2.2)

- Problems with doing these operations with the image itself:
  - Lets imagine we want to do both these operations with the image itself X
  - Let our image be of size N
  - In the authentication use case we need only to compare 2 images, each of size N => O(2*N) and O(N) (time/memory)
  - In the identification use case we need to compare to J albums (let's assume each album has 1 image), every image of size N => O(J*N) and O(N*J) (time/memory)
  - Euclidean distance is a not a good measure of "likeness" in an image (not robust to geometrical operations)

(2.2)

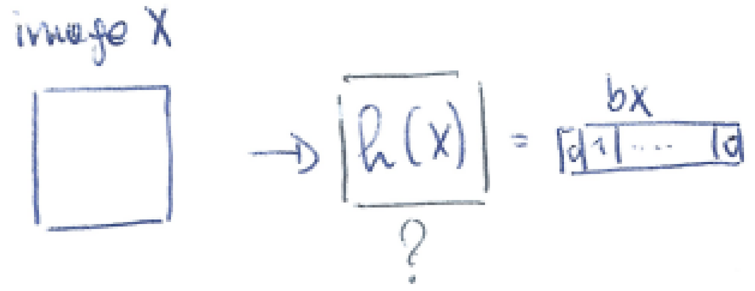
- If we instead compare the binary representation of each image bx, we get a certain number of advantages:

  - Speed and memory, as we can replace N by L (N >> L)
  - Vectors are binary, one can compute hamming distance instead of Euclidean distance
  - Secure in the event of data breaches (can’t recreate X from bx)
  - Robust to geometrical operations, compression

  - Follows known distribution (question 2.4)

(2.3) Explain the construction of content fingerprinting function based on random projections and binarization

- Recall:
  - Noted: h(X) = bx
    1. Input (X) is a binary string of any length (N)
    2. Output (bx) is a binary string of a given length (L)



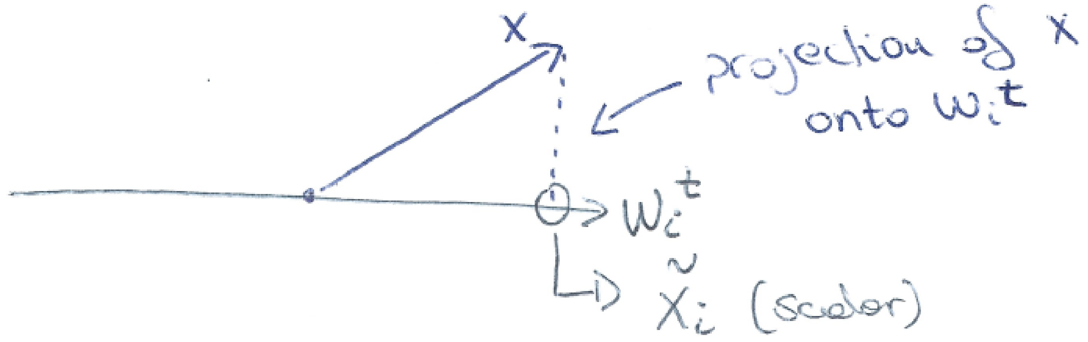
(2.3)

- But what is h() doing exactly ?
    1. Vectorize input X
    2. Compute matrix W
    3. Compute: $X_i^{\sim} = W_i^t * X$
    4. Turn $X_i^{\sim}$ into a 1 or a 0 (binarization)

$$\begin{bmatrix} | \\ \underline{bx} \\ | \end{bmatrix}_{L}^{1} \approx \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_L \end{bmatrix}_{L}^{1} = \begin{bmatrix} w_1^t x \\ w_2^t x \\ \vdots \\ w_L^t x \end{bmatrix}_{L}^{1} = \begin{bmatrix} — & w_1^t & — \\ — & w_2^t & — \\ & \vdots & \\ — & w_L^t & — \end{bmatrix}_{L}^{N} \begin{bmatrix} | \\ \underline{X} \\ | \end{bmatrix}_{N}^{1}$$

(2.3)

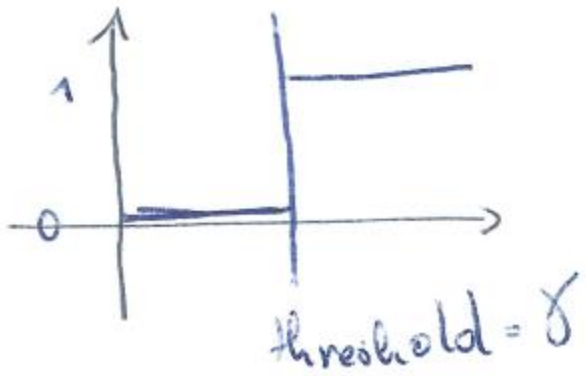
- Projection:



$$\tilde{X}_i = \sum_{j=0}^{N} X_j \cdot W_{ij}$$

- Binarization:

$$\begin{tikzcd} & 0 \\ \tilde{X}_i \arrow[ur] \arrow[dr] & ? \\ & 1 \end{tikzcd}$$



if $\tilde{X}_i \geq \gamma \rightsquigarrow 1$

if $\tilde{X}_i < \gamma \rightsquigarrow 0$

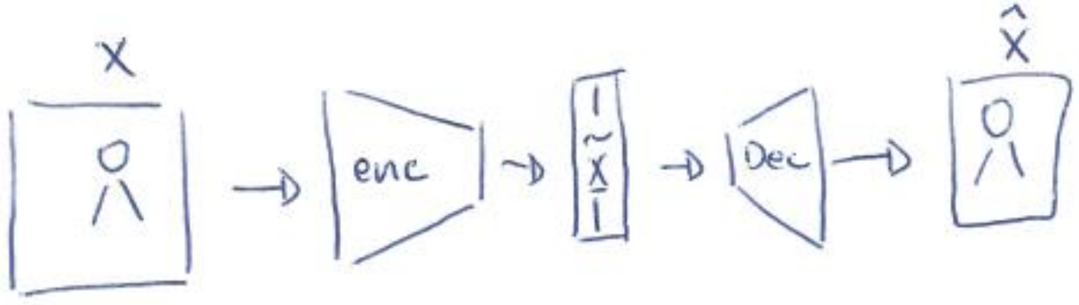
(2.3)

- How do we compute matrix W ?

  There are 2 possibilities, it is either handcrafted (HC), or machine learning (ML)

- HC uses a secret key k, that it uses as a seed to generate random values for W, whoever has access to the key k, can compute the same matrix W

- ML uses machine learning algorithms to create the perfect matrix W, no key required



Goal : $\min \|X - \hat{X}\|_2^2$

(2.3)

- One question remains, which FP model is better ?
  - ➢ ML offers more robustness to geometrical operations/compression on X
  - ➢ ML needs training time to create a perfect encoder
  - ➢ ML is fully data dependant, meaning an attacker that has access to the data, can train his own model, and have a decoder

- Both can be used, depending on the application one might be more appropriate!

(2.4) Explain the statistics of coefficients under random projections used in content fingerprinting. Properties.

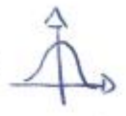
1. Vectorize input X
2. Compute matrix W
3. Compute: $\tilde{X_i} = W_i^t * X$
4. Turn $\tilde{X_i}$ into a 1 or a 0 (binarization)

$$\begin{bmatrix} | \\ \underline{bx} \\ | \end{bmatrix}^{1}_{L} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_L \end{bmatrix}^{1}_{L} = \begin{bmatrix} w_1^t x \\ w_2^t x \\ \vdots \\ w_L^t x \end{bmatrix}^{1}_{L} = \begin{bmatrix} — & w_1^t & — \\ — & w_2^t & — \\ & \vdots & \\ — & w_L^t & — \end{bmatrix}^{N}_{L} \begin{bmatrix} | \\ \underline{X} \\ | \end{bmatrix}^{1}_{N}$$

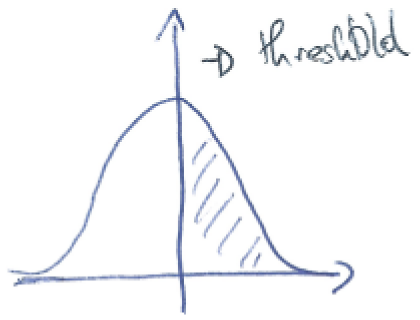
(2.4)

- It is impossible to know the statistical distribution of X
- However it is very possible to know the statistical distribution of bx!
- Let’s do it:
  - First of we have to understand that bx_i is statistically independent from bx_{i+1}, even though x_i and x_{i+1} are dependant, this is because the multiplication => $W_i^t * X$, will make them independent
  - When we add multiple independent variables, the result is a gaussian of mean 0!

$$\tilde{X}_i = W_i^t X = W_{i1}^t X_1 + W_{i2}^t X_2 + \dots + W_{in}^t X_n$$

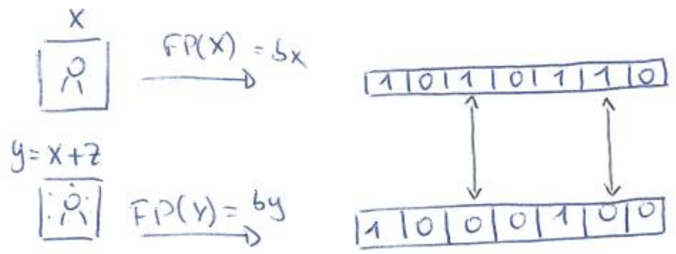
$$\text{if } N \to \infty \Rightarrow p(\tilde{x}_i) \approx$$ 

  - Because $x_i^{\sim}$ is gaussian of mean 0, it means that bx is of maximum entropy (if we choose the threshold correctly)!



(2.4)

- Because bx_i and bx_{i+1}, we can compute the probability of error Pb, between two different binary strings



$$Pb = \frac{1}{L} \sum_{i=1}^{L} \underbrace{Pr[bx_i \neq by_i]}_{\substack{1 \text{ if different} \\ 0 \text{ if same}}}$$

- There are L independent events, and at each attempt there is a probability Pb of flip. This is a Binomial distribution!
- From here we can compute expectancy and variance

$$E[H^d(bx, by)] = L \cdot Pb$$

$$Var(H^d(bx, by)) = L \cdot Pb \cdot (1 - Pb)$$

- With L large enough, our binomial distribution will be gaussian shaped (symmetrical). Which also means we can easily compute Pm and Pfa

(2.4)

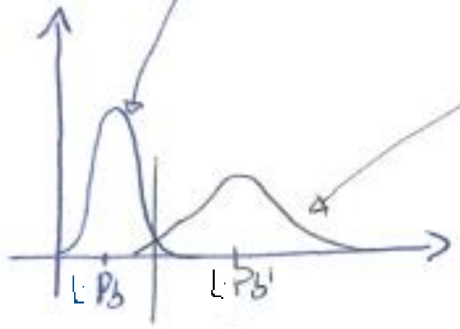
b_x b_y $b_{x'}$ $b_{y'}$

$$H^d(b_x, b_y) \ll H^d(b_{x'}, b_{y'})$$

$$P_b \ll P_b'$$

$$\Pr\left[H^d(b_x, b_y) \le \gamma L\right] \to 1 \qquad \Pr\left[H_0^d(b_{x'}, b_{y'}) \le \gamma L\right] \to 0$$

$$H^d(b_x, b_y) \sim \underbrace{\beta(L, P_b)} \qquad H^d(b_{x'}, b_{y'}) \sim \underbrace{\beta(L, P_b')}$$

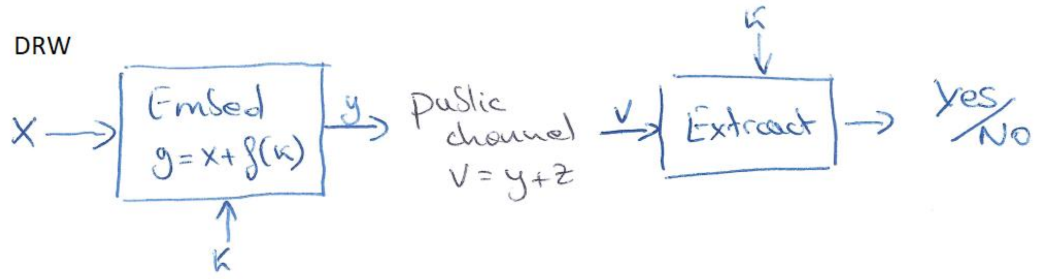


↳ threshold of acceptance $\to \gamma \cdot L$

$\Rightarrow$ P_{miss}, P_{fa} known!

(2.5) Explain the difference between the sufficient statistics in digital watermarking (linear cross correlation) and Hamming distance? What are the implications of these differences?

- Linear cross correlation:
  - Used to detect the watermark in a media file
  - Block diagram:



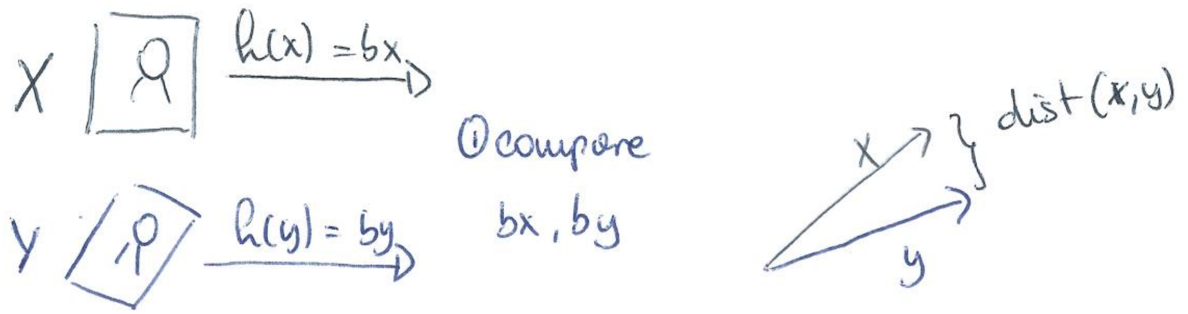
  - Cross correlation used to answer yes/no

1. Compute w using k
2. Compute $x^{\wedge} = denoise(v)$
3. Compute $w^{\wedge} = v - x^{\wedge}$
4. Compute $\rho = \frac{1}{N} \sum w^{\wedge}[i] * w[i], \;\; i = 0..N$

➢ If$\rho \geq \gamma \rightarrow$ accept, γ the threshold

(2.5)

- Hamming distance:
  - Used to decide if two FingerPrints are from close inputs



- Hamming distance used to answer yes/no

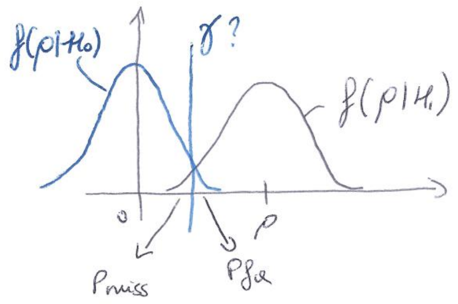
1. Compute $y = x + z$ $\rightarrow z$ is any distortion
2. Compute $b_x = FP(x)$
3. Compute $b_y = FP(y)$
4. Compute $b = D^h(b_x, b_y)$

➢ If $b < \gamma \rightarrow$ accept, γ the threshold

(2.5)

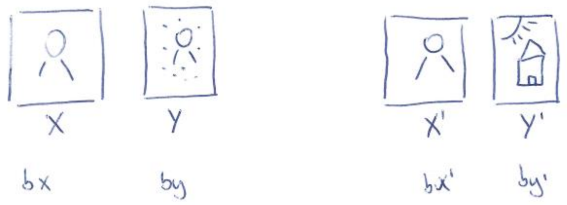
- Cross correlation result follows a gaussian distribution of mean $\mu = 0$
- Hamming distance result follows a binomial distribution $B(L, Pb)$

➢ if L is large enough, $B(L, Pb)$ is gaussian shaped!



$$P_m = \int_{-\infty}^{\gamma} f(\rho|H_1)\, d\rho$$

$$P_{fa} = \int_{\gamma}^{+\infty} f(\rho|H_0)\, d\rho$$

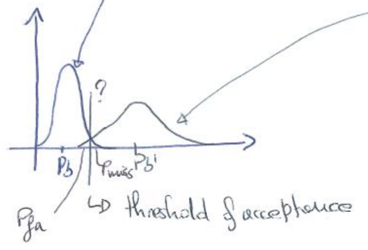


$$H^d(bx, by) \ll H^d(bx', by')$$

$$Pb \ll Pb'$$

$$\Pr[H^d(bx, by) \leq \gamma L] \to 1 \qquad \Pr[H_0^d(bx', by') \leq \gamma L] \to 0$$

$$H^d(bx, by) \sim B(L, Pb) \qquad H^d(bx', by') \sim B(L, Pb')$$



(2.6) Explain the main requirements to privacy protection. Provide several examples of privacy protection schemes.

- Lets assume you are a phone developer
- You have a face unlocking functionality
- It takes pictures of the users face to authenticate them

➢ Should you store the users real face ?

- You are always at risk of data breaches !

(2.6)

- By having your users images/photos, someone could:
  1. Create silicone masks that are real enough to trick computer vision
  2. Sell private information to insurances (by analysing face details freckles/irises)
  3. …

- The goal is not needing to store real data, but still offer the possibility of face unlocking

- To achieve this, we look at cryptographic hashing functions!
  1. They can take as input any data length
  2. Output is smaller than input (easier to store)
  3. From the output you cant compute input (safe in event of data breaches)

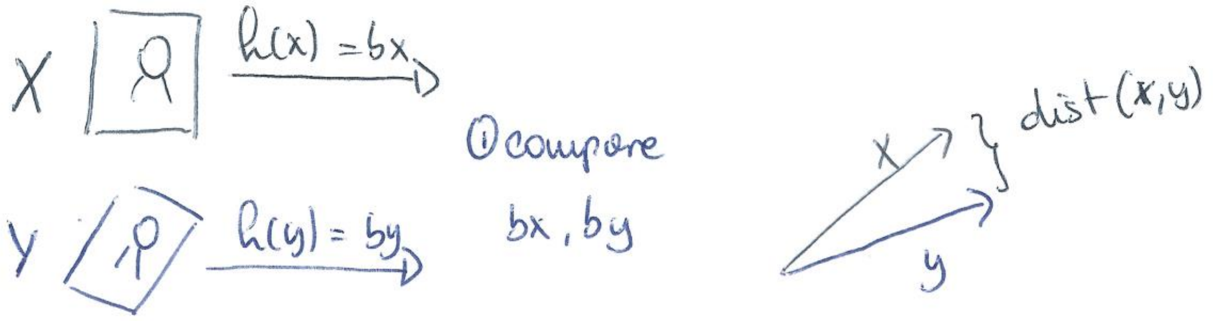
(2.6)

- The only problem is that if you move your face in the slightest, the output hash will be completely different!

- So you wont be able to compare two ashes to decide whether or not it is the same person in two different pictures

➢ Solution -> use fingerprinting hashes:
   1. All the same properties as cryptographic hashing
   2. But 2 slightly different images, will have the same or almost the same fingerprint hash

- With this method, you get the best of both worlds, security while keeping costumers happy


X
h(x) = bx
Y
h(y) = by
① compare
bx , by
x
y
dist(x,y)