Bildverschlüsselung mit Matlab

Bericht

Berger Marco
Pollari Andy
Programmierung in Matlab/Octave

14. Januar 2015

1 Einleitung

In dieser Arbeit befassen wir uns mit der Anwendung verschiedener Verschlüsselungsalgorithmen angewandt auf Bilder implementiert in Matlab.
Es ist zu erwähnen, dass es grundsätzlich zwei verschiedene Verschlüsselungsverfahren gibt:

- Die symmetrische Verschlüsselung
- Die asymmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung wird mit einem Schlüssel verschlüsselt und mit demselben auch wieder entschlüsselt. Bei der asymmetrischen Verchlüsselung hingegen gibt es zwei verschiedene Schlüssel: Einen öffentlichen Schlüssel zum verschlüsseln und einen privaten Schlüssel zum entschlüsseln.

Es gibt verschiedene asymmetrische Verschlüsselungsverfahren wie RSA, Merkle-Hellmann, RSA, ...
Auch bei den symetrischen Verschlüsselungsverfahren gibt es verschiedene wie DES, AES, One-Time-Pad, ...
Im Rahmen dieser Arbeit konzentrieren wir uns bei der symmetrische Verschlüsselung auf das *One-Time-Pad* und bei den asymmetrisch Verschlüsselungsverfahren auf RSA.

In dieser Arbeit haben wir festgestellt, dass sich Matlab nur bedingt eignet, um Bilder zu Verschlüsseln. Für die symmetrische Verschlüsselung stiessen wir auf keine grösseren Probleme. Bei der asymmetrischen Verschlüsselung trafen wir auf ein grösseres Problem bezüglich Primzahlen. Dieses Problem erläutern wir später im Kapitel 4 *Ergebnisse, Resultate*

2 Grundlagen

In dieser Arbeit beschränken wir uns auf folgende Verschlüsselungsverfahren:

- One Time Pad, als symmetrische Verschlüsselung
- Rivest, Shamir und Adleman (RSA), als asymmetrische Verschlüsselung.

Daher beschränken wir uns ausschliesslich auf diese beiden Verfahren.

2.1 One-Time-Pad

Beim One-Time-Pad haben wir einen Keystream der aus random Bits besteht. Dieser Keystream muss mindestens so viele Bits lang sein, wie die zu verschlüsselnde Nachricht selbst. In unserem Projekt ist die zu verschlüsselnde Nachricht ein JPG-Bild.
Die Idee beim One-Time-Pad ist, dass der Keystream nur einmal zum ver- respektive entschlüsseln verwendet wird.

Die Verschlüsselung beim One-Time-Pad wird realisiert, indem das Bit an der Position i des Bildes m mit dem i-ten Bit des Keystreams r XOR verknüpft wird.

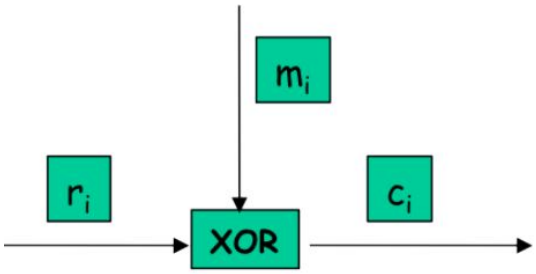


Abbildung 1: One-Time-Pad Encryption

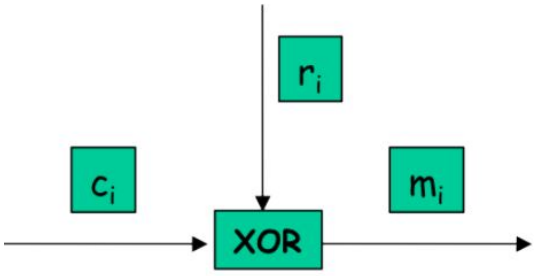


Abbildung 2: One-Time-Pad Decryption

2.2 Rivest, Shamir und Adleman (RSA)

RSA ist ein asymmetrisches kryptographisches Verfahren. Es wird zur Verschlüsselung aber auch für digitalen Signaturen verwendet. In dieser Arbeit befassen wir uns ausschliesslich mit dem Verschlüsselungsverfahren von RSA.

RSA verwendet ein Schlüsselpaar bestehend aus einen privaten Schlüssel (Private Key) und einem öffentlichen Schlüssel (Public Key). Den Private Key verwendet man dabei um Daten zu entschlüsseln, die mit dem Public Key verschlüsselt worden sind.

2.2.1 Schlüssel Generierung

Der Public Key ist ein Zahlenpaar (e, N), und der Private Key ist auch ein Zahlenpaar (d, N). e des Public Keys wird auch den Verschlüsselungsexponenten genannt. d des Private Keys wird auch den Entschlüsselungsexponenten genannt. Bei beiden Zahlenpaaren ist N gleich und wird *RSA Modul* genannt. Eine kurze grobe Beschreibung, wie die oben genannten Schlüssel generiert werden:

- Die Primzahlen p und q zufällig wählen, $p \neq q$
- Den RSA-Modul N berechnen

$$N = p \cdot q \tag{1}$$

- $\varphi(N)$ berechnen

$$\varphi(N) = (p-1) \cdot (q-1) \tag{2}$$

- e wählen, welches teilerfremd von $\varphi(N)$ ist
- d wählen, wobei gilt

$$e \cdot d \equiv_N mod \varphi(N) \tag{3}$$

  d ist also das multiplikativ Inverse Element von e im Bezug auf $\varphi(N)$

2.2.2 Verschlüsselung / Entschlüsselung

Möchten wir eine Nachricht m verschlüsseln, so wird die Nachricht m mit dem Verschlüsselungsexponenten e potenziert. Man erhält so den Ciphertext c (Geheimtext).

$$c \equiv_N m^e \tag{4}$$

Um den Ciphertext c zu entschlüsseln, muss c mit dem Entschlüsselungsexponenten d potenziert werden. So erhalten wir wieder die Ursprungsnachricht m.

$$m \equiv_N c^d \tag{5}$$

3 Vorgehen, Methoden Analysen

Da wir die Grundlagen der zu verwendenden Verschlüsselungen bereits durch unsere Studium-Vertiefung IT-Security hatten, konnten wir uns schnell der Implementierung widmen.

3.1 Implementierung der symmetrischen Verschlüsselung

Wie bereits erwähnt haben wir uns für eine One Time Pad Verschlüsselung entschieden. Dabei wird ein Bild in eine Matrix M_M gelesen. Für das One Time Pad erstellen daraufhin eine Matrix M_R mit Zufallswerten und exakt der Grösse, des eingelesenen Bildes. Die beiden Matrizen werden XOR miteinander verknüpft, was eine neue Matrix M_C ergibt und dem verschlüsselten Bild entspricht. Nachträglich haben wir die Zeitmessung eingebaut, um Auswertungen machen zu können.

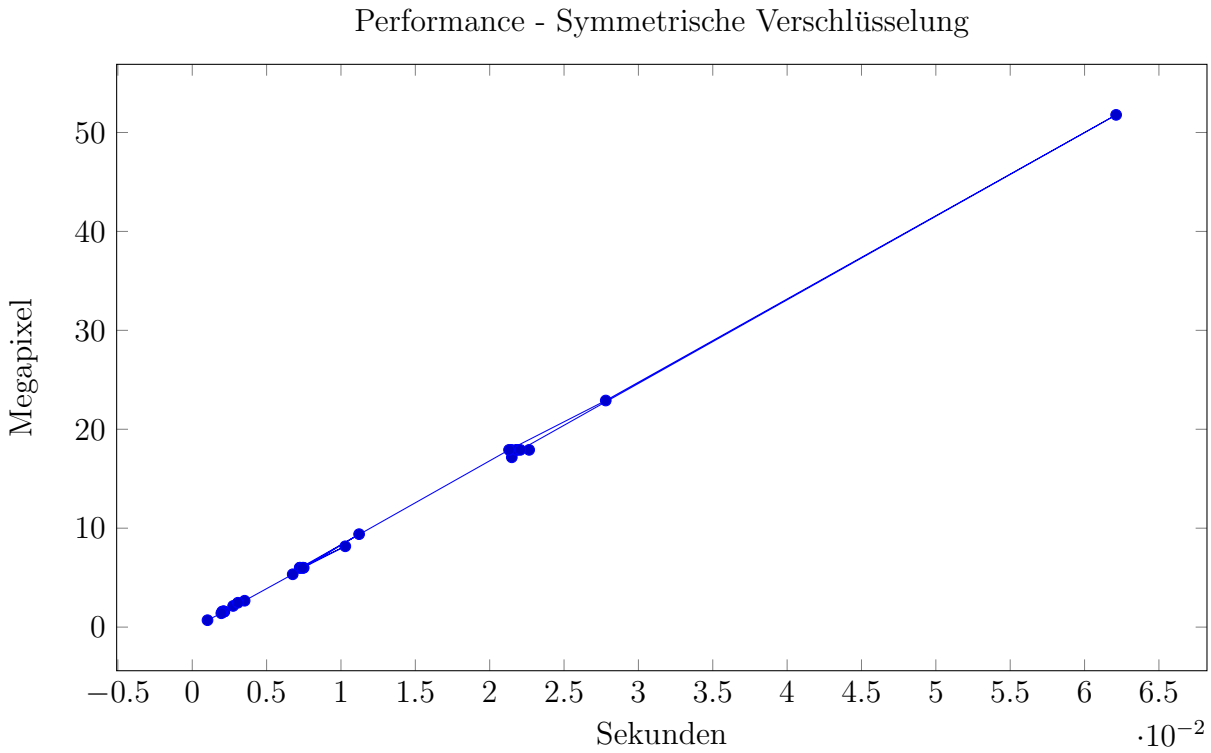
3.2 Implementierung der asymmetrischen Verschlüsselung

Bei der asymmetrischen Verschlüsselung haben wir uns für die RSA Verschlüsselung entschieden. Die Implementiertung erfolgte erst ziemlich genau so, wie im Grundlagen Kapitel 2.2 beschrieben: Erst die Schlüsselgenerierung und anschliessend wurde jeder Wert des Bildes mit den Bilder ver- und entschlüsselt. Bei der RSA Implementierung stiessen wir auf einige Probleme, die wir im Kapitel 4.2.3 erläutern. Nach der vollständigen Implementierung der Ver- resp. Entschlüsselung, wurden auch hier noch Zeitmessungen eingebaut, um Auswertungen machen zu können.

4 Ergebnisse, Resultate

4.1 Symmetrische Verschlüsselung

4.1.1 Performance bei der symmetrischen Verschlüsselung



Unserer Meinung nach ist die Performance bei der symmetrischer Verschlüsselung sehr akzeptabel. Die obere Grafik zeigt, dass die Verschlüsselungsdauer wie erwartet linear zur Anzahl Bildpunkten ist. Um ein Bild von 50 Megapixel zu verschlüsseln braucht unser Matlab Programm 0.06s. Die Verschlüsselungsdauer ist mit der Entschlüsselungsdauer identisch, da die gleichen Operationen (XOR) durchgeführt werden. Bezüglich der Performance eignet sich Matlab unserer Meinung sehr gut für eine symmetrische Verschlüsselung mit dem One-Time-Pad.

4.1.2 Wiedererkennbarkeit des Bildes

Mehrere Versuche zeigten, dass das Originalbild nach der Verschlüsselung in keiner Weise wiedererkennbar ist. Auch in diesem Punkt, überzeugt die symmetrische Verschlüsselung mit dem One-Time-Pad. Die Abbildung 3 zeigt ein Beispiel des Resultates.

Abbildung 3: One Time Pad - Verschlüsselung

4.1.3 Verschlüsseltes Bild Speichern und anzeigen

Das verschlüsselte Bild kann als valides JPG Bild abgespeichert werden und in einem gewöhnlichen Bildbetrachtungsprogramm interpretiert werden.

4.2 Asymmetrische Verschlüsselung

4.2.1 Performance bei der asymmetrischen Verschlüsselung

Bei der asymmetrischen Verschlüsselung lässt die Performance zu wünschen übrig. Bei der Verschlüsselung eines 50 Megapixel zu verschlüsseln, brauchten wir auf unseren Workstations durchschnittlich 337 Sekunden, also 5.6 Minuten. Dies ist über 5000x langsamer als bei unserer symetrischen Verschlüsselung.

4.2.2 Wiedererkennbarkeit des Bildes

Bei einigen Bildern konnte man beim verschlüsselten Bild noch Umrisse, Konturen oder grössere Flächen wiedererkennen. Somit wurden die Bilder nicht immer völlig verschleiert und teilweise waren sie wiedererkennbar.

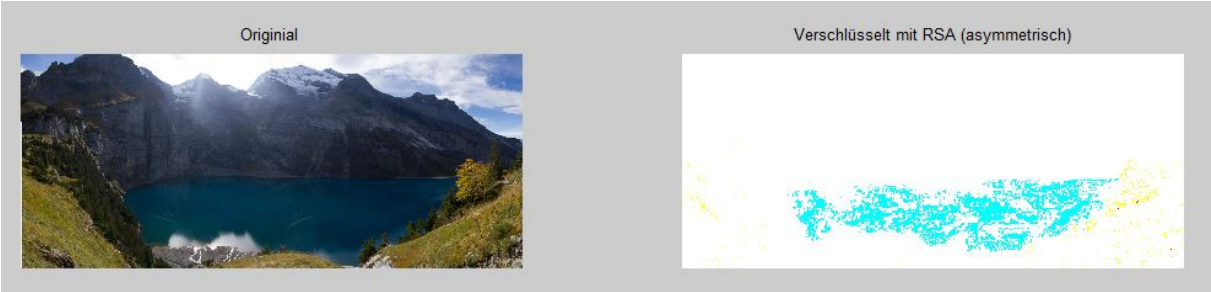


Abbildung 4: RSA - Verschlüsselung

4.2.3 Problem bei der RSA Verschlüsselung

Ungültiges Bild Wie man in der Abbildung 4 sieht, ist der grösste Teil des Bildes weiss. Grund dafür ist, dass wir mit JPG Bilder von 8-Bit arbeiteten. Mit einer Farbtiefe von 8-Bit kann man pro Farbe 255 verschiedene Werte darstellen. Grund dafür ist, dass wir mit JPG Bildern von 8-Bit Farbtiefe arbeiteten. Mit einer Farbtiefe von 8-Bit kann man pro Farbe 256 verschiedene Werte darstellen.

Da wir nun die RGB Werte von jedem Pixel verschlüsseln, kann es also für die einzelnen RGB Werte grössere Farbwerte als 255 geben, was eigentlich zu Farben führt, die ausserhalb des 8-Bit Spektrums liegen.

Der Bildbetrachter von Abbildung 3 interpretiert diese double RGB-Werte nun als Weiss. Es gehen also nicht Informationen verloren, wie es vielleicht auf den ersten Blick schient, sondern viele Werte liegen ausserhalb der 8-Bit Tiefe, welche dann weiss angezeigt werden. Somit entstehen neu double-Werte die Matlab dennoch als Bild darstellen kann. Werte die nun ausserhalb der 8-Bit Farbtiefen liegen, werden als weiss interpretiert. Matlab kann nun mit der RSA Modulo-Operation das Ausgangsbild wieder korrekt zurückrechnen. Allerdings ist es nicht möglich, dieses Bild direkt interpretierbar abzuspeichern (jedenfalls ohne zusätzliche mapping Operationen). Es wäre möglich, mit Bilder von einer höheren Farbtiefe zu arbeiten wie bspw. 16-Bit Bildern. Das oben genannte Problem wäre somit immer noch bestehend, einfach erst mit grösseren Zahlen.

Problem Performance Die Performance mit der RSA Verschlüsselung ist schlecht. Im Allgemeinen eignet sich die asymmetrische Verschlüsselung nicht bei grossen Dateien.

Üblicherweise wird ein hybrides Verschlüsselungsverfahren verwendet indem ein symmetrischer Schlüssel asymmetrisch verschlüsselt wird. Auch unsere Performance Tests zeigten, dass die Performance bei der RSA-Verschlüsselung sehr schlecht ist (rund 1000x langsamer) im Vergleich zu der symmetrischen Verschlüsselung.

Problem Pixel für Pixel Verschlüsselung Jedes Pixel besteht aus drei Farben: Rot, Grün und Blau. Bei unseren Tests haben wir nur mit JPG Bildern mit einer Farbtiefe von 8-Bit gearbeitet. Das heisst dass wir für die Farben Rot, Grün, Blau je 255 verschiedene Mögliche Werte haben können. Unser asymmetrisches Verschlüsselungsverfahren iteriert durch jedes Pixel P_n und verschlüsselt in Jedem Pixel die Farben.

$$Original : P_n(R, G, B) \tag{6}$$
$$Encrypted : P'_n(enc(R), enc(G), enc(B)) \tag{7}$$

Durch das RSA Verschlüsselungsverfahren kann es dann aber für $enc(R|G|B)$ Werte geben die nicht mehr im 8-Bit Bereich sind (Problem oben beschrieben). Ein anderes Problem ist aber auch, dass der selbe RGB-Wert verschlüsselt auch immer den zum selben verschlüsselten Wert führt. Dies erklärt auch die hellblaue Fläche im verschlüsselten Bild in der Abbildung 4

Keine grossen Primzahlen Wir wussten bereits im Vornhinein, dass wir für Matlab nur relativ kleine Primzahlen nutzen können. Mit einer Bruteforce Attacke könnte man also unsere mit Matlab asymmetrisch verschlüsselten Bilder in absehbarer Zeit „entschlüsseln“.

5 Ahnang

5.1 Code

5.1.1 Symmetrische Verschlüsselung

```
% Declare image to load
% files = dir('images/large-2.jpg');

% Load all image files
files = dir('images/*.jpg');

% initialize matrixes for our csv report
time = [];
imgName = [];
width = [];
height = [];
megapixel = [];

% iterate all files
for file = files'
    filename = file.name

    % concat the image path
    imagePath = strcat('images/', filename);
    total_runtime = tic;
    % load image
    my_image = imread(imagePath, 'jpg');
    % get image dimensions
    my_image_size = size(my_image);
    y = my_image_size(1);
    x = my_image_size(2);
    z = my_image_size(3);

    % generate random matrix for the one time pad
    randomMatrix = randi([0,255], y, x, z);
    randomMatrix = uint8(randomMatrix);

    % encrypt image
    image_encrypted = my_image;
    enc_time = tic;
    image_encrypted = encData(my_image, randomMatrix);
    enc_time = toc(enc_time);
    % image_decrypted = encData(image_encrypted, randomMatrix);

    % create statistics for the table
    time = cat(1, time, [enc_time]); % time in seconds
    width = cat(1, width, [x]);
    height = cat(1, height, [y]);
    megapixel = cat(1, megapixel, [(x*y)/10^6]);
    imgName = [imgName; {filename}];

    %figure()
    subplot(1,3,1)
    imshow(my_image,[])
    title('Original image')
    subplot(1,3,2)
```

```
    imshow(image_encrypted,[])
    title('Encrypted image')
    % subplot(1,3,3)

  % activate next 2 lines to show the decrypted image
    %imshow(image_decrypted,[])
    %title('Decrypted Image')
end

% Create table for Report and save it to a csv file
T = table(imgName,width, height,megapixel, time);
writetable(T,'performance-symmetric-enc.csv');
```

matlab-enc-symetric.m

```
function retval = encData(ColorVal, randomVal)
    encrypted = bitxor(ColorVal, randomVal);
    retval = im2uint8(encrypted);
end
```

encData.m

5.1.2 Asymmetrische Verschlüsselung

```
files = dir('images/*.jpg');

% initialize matrixes for report
time = [];
imgName = [];
width = [];
height = [];
megapixel = [];

% iterate through all the files in images/ folder
for file = files'
    filename = file.name

    % read image
    imagePath = strcat('images/', filename);
    total_runtime = tic;
    % read image
    clean_image = imread(imagePath, 'jpg');
    % get size
    my_image_size = size(clean_image);
    y = my_image_size(1);
    x = my_image_size(2);
    z = my_image_size(3);

    enc_time = tic;

    % generate a p and q which are not the same and where the result of the
    % multiplication is bigger or equal to 255
    n=0;
    while((n < 255) || (p == q))
        p = randseed(randi(1000),1,1,1,25);
        q = randseed(randi(1000),1,1,1,25);
        n = p * q;
    end

    % calculate the totient
    t = (p-1) * (q-1);

    % generate a public key e_ which is divisor-TODO with 't'
    e_ = randseed(randi(1000),1,1,1,t);
    while(gcd(e_, t) ~= 1)
        e_ = randseed(randi(1000),1,1,1,t);
    end
    % calculate the multiplicative inverse of e_, so that d * e [=] 1 mod t
    % which results in the private key 'd'
    d = multiplicative_inverse(e_,t);

    % Declare image to load
    % read in jpg image
    % clean_image = imread('island.jpg','jpg');


    % encrypt image with RSA
    image_encrypted = clean_image;
    image_encrypted = encDataRSA(clean_image, e_, n);
```

```
        enc_time = toc(enc_time);
    % decrypt image with RSA
    image_decrypted = decDataRSA(image_encrypted, d,n);

    % create statistics for the table
    time = cat(1, time, [enc_time]); % time in seconds
    width = cat(1, width, [x]);
    height = cat(1, height, [y]);
    megapixel = cat(1, megapixel, [(x*y)/10^6]);
    imgName = [imgName; {filename}];

    % plot images
    figure()
    subplot(1,3,1);
    imshow(clean_image,[])
    title('Originial')

    subplot(1,3,2);
    imshow(image_encrypted,[])
    title('Verschlsselt mit RSA (asymmetrisch)')

    subplot(1,3,3);
    imshow(image_decrypted,[])
    title('Entschlsselt')
end

T = table(imgName,width, height,megapixel, time);
writetable(T,'performance-asymmetric-enc.csv');
```

matlab-enc-assymetric.m

```
function retval = encDataRSA(ColorVal, e, n)
    ColorVal = double(ColorVal);

    cipher=1;
    % workaround for calculating  the power of e
    for k=1:e
        cipher=mod(cipher.*ColorVal,n);
    end
    retval = cipher;
end
```

encDataRSA.m