Análise Visual de Cifras: Vinégere, Affine Cipher e TEA

Universidade Estadual de Campinas

Felipe Santos Oliveira
RA 119383

28 de maio de 2015

1 Introdução

Este projeto teve como objetivo a implementação de de três algorítimos de encriptação com o objetivo de cifrar imagens no formato PPMe e analisar visualmente os efeitos das cifragens. Os algorítimos de encriptação usados foram *Vinégere*, *Affine* e *Tiny Encryption Algorithm* (TEA). A seguir, mostro uma breve explicação de cada uma das cifras.

2 Cifras

2.1 Vinégere

A cifra Vinégere é um método de encriptação que consiste em diversas cifras de César usadas em sequência com valores de *shift diferentes*. Algebricamente, para uma mensagem $\mathcal{M}$, uma chave $\mathcal{K}$, a função de encriptação $\mathcal{E}$ é definida por:

$$\mathcal{C}_i = \mathcal{E}_\mathcal{K}(\mathcal{M}_i) = (\mathcal{M}_i + \mathcal{K}_i) \bmod 26$$

E a decriptação $\mathcal{D}_i$ é dada por:

$$\mathcal{M}_i = \mathcal{D}_\mathcal{K}(\mathcal{C}_i) = (\mathcal{C}_i - \mathcal{K}_i) \bmod 26$$

onde $\mathcal{M} = \mathcal{M}_1||...||\mathcal{M}_n$ é a mensagem, $\mathcal{C} = \mathcal{C}_1||...||\mathcal{C}_n$ é a mensagem cifrada e $\mathcal{K} = \mathcal{K}_1||...||\mathcal{K}_n$ é a chave obtida através da repetição da palavra-chave n/m vezes, onde m é o tamanho da palavra chave.

2.2 *Affine*

A *Affine Cipher* é uma cifra de fluxo que é uma extensão da cifra de César, assim como a cifra de Vinégere. Na *affine cipher*, as letras de um alfabeto de tamanho m são mapeadas no intervalo $[0, ..., m-1]$. É usada, então, álgebra modular para o inteiro correspondente à mensagem pura em outro inteiro correspondente à mensagem cifrada. Algebricamente, a função de encriptação $\mathcal{E}$ é definida por:

$$\mathcal{C}_i = \mathcal{E}_\mathcal{K}(\mathcal{M}_i) = (a\mathcal{M}_i + b) \bmod 26$$

E a função de decriptação é definida por:

$$\mathcal{M}_i = \mathcal{D}_\mathcal{K}(\mathcal{C}_i) = a^{-1}(\mathcal{M}_i - b) \bmod 26$$

onde onde $\mathcal{M} = \mathcal{M}_1||...||\mathcal{M}_n$ é a mensagem, $\mathcal{C} = \mathcal{C}_1||...||\mathcal{C}_n$ é a mensagem cifrada, $\mathcal{K} = \{a, b\}$, $a, b \in \mathbb{N}$ e a^{-1} sendo o inverso modular de a.

2.3 TEA

O *Tiny Encryption Algorithm* é um algorítimo desenvolvido por dois pesquisadores na Universidade de Cambridge em 1994. Ela é uma crifra em blocos que usa uma rede de Feistel como estrutura de encriptação, blocos de 64 bits e uma chave de 128 bits em 32 *rounds*. Apesar de ser extremamente rápido, o algorítimo possui fraquezas que diminuem sua força de 128 para 126 bits e é ruim como função hash. Abaixo encrontra-se uma implementação do algorítimo em C para referência:

```
void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint32_t delta=0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i < 32; i++) {                       /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;  /* set up */
    uint32_t delta=0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

3 Implementação

Foi realizada a implementação somente das cifras TEA e *Affine* para esse projeto, ambas na linguagem Python, e nos dado uma implementação em C da cifra de Vinégere. A implementação da cifra *Affine* usou funções auxiliares para cálculo do menor divisor comum e inverso modular, e operou diretamente emcima dos pixels da imagem.

A implementação do TEA, dado que a linguagem escolhida foi Python, foi mais trabalhosa, dado que o TEA realiza operações binárias em blocos de 64 bits, o que levou a necessidade de realizar uma manipulação dos dados antes e depois da execução das funções de encriptação e decriptação. Esse tratamento de dados consiste em transformar uma lista de listas em uma lista de inteiros (função `flatten/unflatten`) e um empacotamento dos pixels em inteiros (função `packbytes/unpackbytes`). Para isso, cada conjunto de 4 pixels era convertido em um inteiro de 32 bits, e dois desses inteiros consitituiam um bloco. Tratamento extra era dado para os dados quando não se era possível completar um bloco, adicionando uma mascara de zeros, que era ignorado para fins de encriptação.

Ainda no TEA, foram implementados dois modos de operação: ECB e CFB. Para o ECB também foi implementado a técnica de *ciphertext stealling*, que compensa quando o último bloco é incompleto. Para isso, encripta-se o penúltimo bloco e então o separa em duas partes: uma cabeça de $b - m$ bits e uma cauda de m bits, onde b é o tamanho do bloco e m o tanto de bits que faltam no último bloco. A cauda é concatenada com o último bloco, e o resultado é encriptado normalmente. O que resulta passa a ser o penúltimo bloco, e a cabeça de $b - m$ bits passa a ser o último bloco. A decriptação funciona simplesmente invertendo o processo.

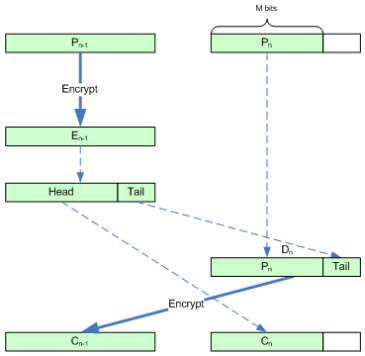

Figura 1: Diagrama mostrando execução do *Ciphertext Stealing* durante a encriptação

4 Resultados

Foram usadas três imagens na execução dos algorítimos: **lena.ppm**, **squares.ppm** e **sample.ppm**:

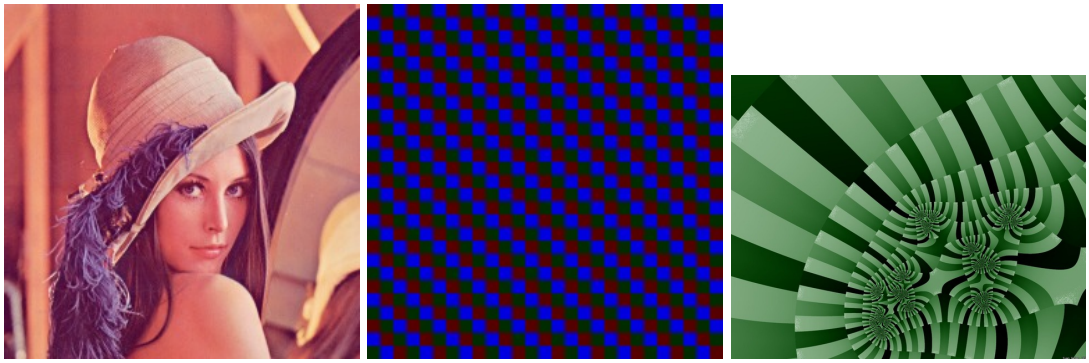

Figura 2: Imagens usadas nos experimentos: lena.ppm, squares.ppm e sample.ppm

4.1 Vinégere

Para o Vinégere, o teste foi executado somente em `lena.ppm` e `squares.ppm`. Foram feitas duas encriptações, uma com um espaço de chave de tamanho 10, escolhidas arbitrariamente, e uma com um espaço de chave de tamanho 2000, geradas usando a função `os.urandom` do Python. Pode-se ver o resultado abaixo:

Figura 3: `lena.ppm` encriptado em espaço de chave tamanho 10 e 2000, respectivamente

Pode-se concluir à partir desse experimento que o Vinégere aparenta maior confusão com o aumento do tamanho da chave utilizada. No entanto, mesmo na imagem com $|\mathcal{K}| = 2000$ pode-se verificar uma leve existência de padrões. Isso se torna evidente com uma imagem com mais padrões, como `squares.ppm`:

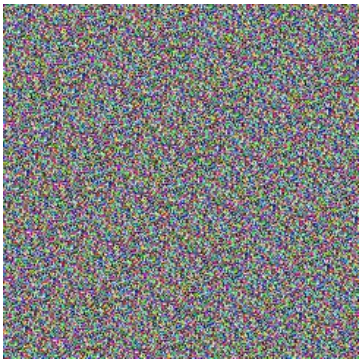

Figura 4: `squares.ppm` encriptado em espaço de chave tamanho 2000

Pode-se ver claramente a existência de padrões nas diagonais da imagem, o que pode facilitar a criptanálise da imagem e extração da chave usada.

4.2 *Affine*

Abaixo pode-se ver o resultado da execução da cifra *Affine* nas três imagens:

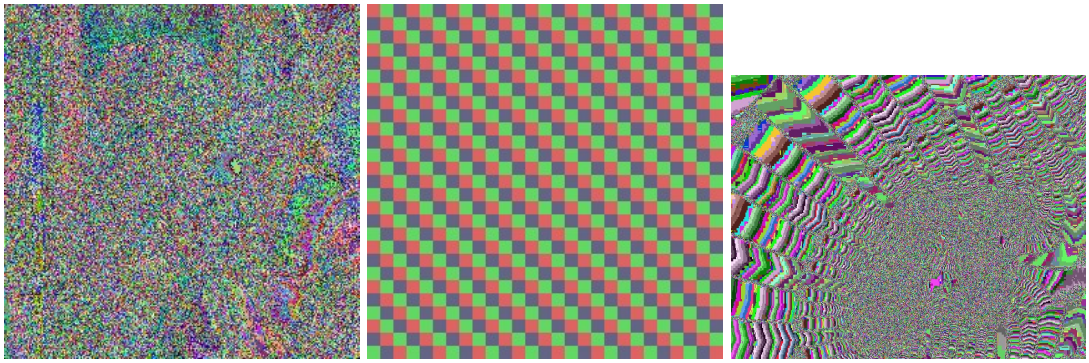

Figura 5: `lena.ppm`, `squares.ppm` e `sample.ppm` cifrados usando a cifra *Affine*

A mesma chave foi usada nas três imagens para fins de experimentação, $a = 55$ e $b = 100$. Em imagens onde não há muitos padrões, como `lena.ppm`, a cifra se mostra mais eficiente, apesar de ainda poder se perceber alguns padrões na imagem. Nas imagens `squares.ppm` e `sample.ppm` o *Affine* se mostra bem menos eficiente, devido ao fato de serem imagens artificiais e com muitos padrões. No caso de `squares.ppm`, os quadrados permanecem evidentes, vazando informações sobre o conteúdo da imagem. Em `sample.ppm`, pode-se ver claramente os padrões e contornos da imagem, apesar de boa parte do conteúdo estar oculto.

4.3 TEA

O TEA se mostrou o mais eficiente dos três algorítimos usados. Foram testados dois modos de execução, ECB e CFB, ambos executados com as mesmas chaves para cada imagem.

4.3.1 Modo ECB

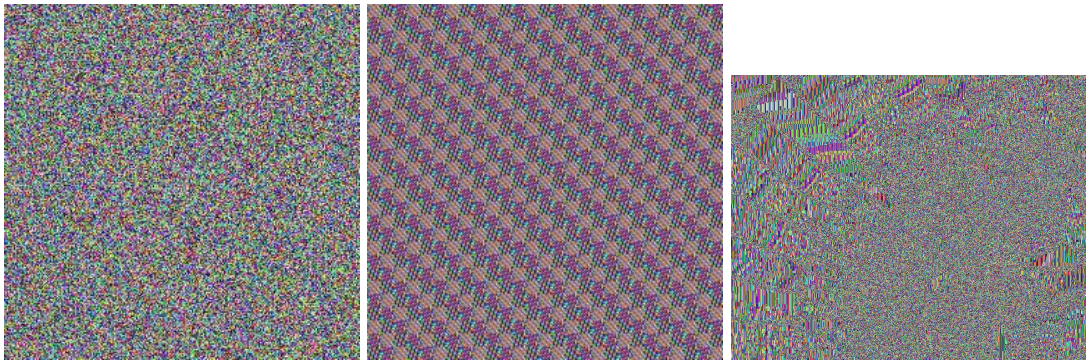

Figura 6: `lena.ppm`, `squares.ppm` e `sample.ppm` cifrados usando o TEA no modo ECB

No modo ECB, como visto acima, o resultado foi parecido com o visto no *Affine*, onde imagens sintéticas tiveram um resultado pior que a foto.

4.3.2 Modo CFB

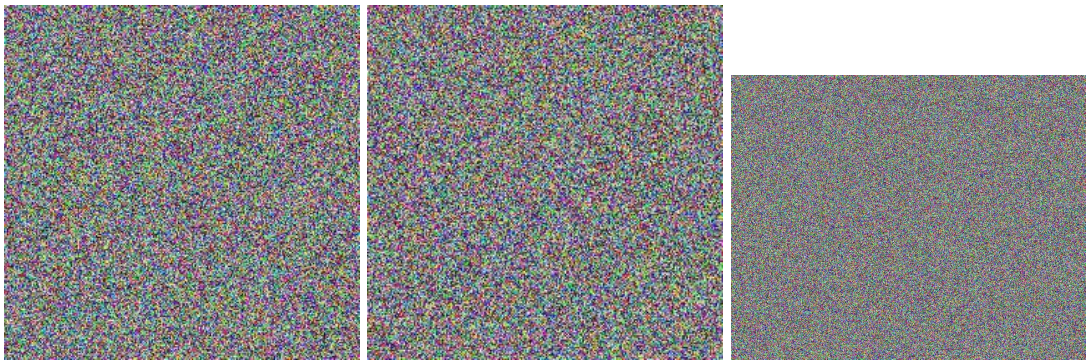

Figura 7: `lena.ppm`, `squares.ppm` e `sample.ppm` cifrados usando o TEA no modo CFB

Este se mostrou o método mais efetivo, completamente ocultando todo o conteúdo e padrões nas imagens. Não é possível extrair qualquer informação nem da foto nem das imagens sintéticas.

5 Conclusão

Pelos experimentos, pudemos constatar que o algorítimo visivelmente mais seguro dentre os analisados foi o TEA no modo CFB. Os outros algorítimos e o modo ECB do TEA se mostraram fracos no sentido de que deixam detalhes como o contorno ou padrões da imagem devido à alta repetição de cores passar para a imagem cifrada. Essa diferença visivelmente grande entre o TEA e os outros algorítimos ocorre principalmente devido ao fato de que a encriptação de cada bloco depender do bloco anterior. Isso adiciona uma camada a mais de segurança, pois não há correspondência direta entre a o bloco na mensagem e o bloco da cifra. O maior problema do TEA durante sua encriptação foi o fato de muitas vezes o último bloco estar incompleto. A implementação do *ciphertext stealing* para o modo ECB se mostrou efetiva e prática, adicionando pouco custo e código. Já no modo CFB um simples *padding* resolve o problema de maneira satisfatória.