O Sistema Merkle-Hellman Knapsack

Bernardo Rodrigues
a79008@alunos.uminho.pt

César Silva
a77518@alunos.uminho.pt

Maria Francisca Fernandes
a72450@alunos.uminho.pt

Universidade do Minho — 5 de Maio de 2019

Resumo

Este documento apresenta os vários passos e considerações feitas para implementação do sistema em questão. Assim como, alguns factos relativos a este.

Conteúdo

Lista de Algoritmos

Capítulo 1

Introdução

Ao contrario do RSA nao da para fazer assinaturas criptograficas - wiki 1976 Diffie Hellman introduzem a ideia de criptografia de chave publica Este trabalho foi desenvolvido no ambito da Unidade Curricular de *Teoria de Números Computacional*. De entre as escolhas possiveis, foi escolhido estudar o sistema *Merkle-Hellman Knapsack*.
Este foi um dos pioneiros da criptografia de chave pública, inventado por **Ralph Merkle** e por **Martin Hellman** em 1978. A ideia por detrás deste sistema é mais simples do que a de sistemas como o *RSA*, assentando no problema (tendo já sido quebrado – meter isto noutro sitio??).

Capítulo 2

Implementação

Ao longo das secções deste capítulo apresentamos os vários algoritmos seguidos para a codificação do sistema. A implementação destes segue em anexo, assim como ficheiros de texto que acompanham o relatório.

2.1 Geração de Permutações

Um dos passos da Geração da Chave Pública - 2.4 - consiste em gerar uma permutação de uma sequência. Para tal utilizamos o algoritmo proposto por **Sandra Sattolo**[1].
Este itera uma lista - seq - a partir do último índice desta - n. Em cada passo calculamos um índice aleatório - j - tal que $1 \leq j < n$ e de seguida trocamos os valores de seq_j e seq_n. Finalmente, terminámos quando $n = 1$.
No nossa implementação usamos listas para guardar as permutações.
A implementacão deste pode ser vista em:

2.2 Geração de um sequência super crescente aleatória

Um dos componentes da Chave Privada é uma sequência, esta é considerada super crescente se:

Definição 1. Consideremos uma sequência de números $b_1, ..., b_n$. Esta diz-se super crescente se:

$$b_i > \sum_{j=1}^{i-1} b_j \text{ para cada } i,\ 2 \leq i \leq n.$$

Como tal, conseguimos deduzir:

$$b_1 + b_2 + ... + b_k < 2 \times b_k$$

Ou seja, precisamos apenas de considerar o último valor gerado para calcular um possivel próximo. Com isto apresentamos o nosso algoritmo.

Algoritmo 1 Geração da sequência super crescente aleatória

Recebe: n - o tamanho da sequência
Devolve: $\{x_1, ..., x_n\}$ - uma sequência super crescente aleatória

k um limite superior aleatóriamente grande
f uma funcão que satisfaz $f(x) > 2 * x$
$x_1 \leftarrow j$ tal que $1 \leq j \leq k$ aleatório
for x_i com $i := 2$ **até** n **do**
    $x_i = f(x_{i-1})$
end for

A implementacão deste algoritmo segue em anexo - ()().

2.3 Geração de Coprimos

Também durante a Geração das Chaves - 2.4 - do sistema temos de considerar dois valores, M e W. Dada um sequência super crescente $\{b_1, ..., b_n\}$, M verifica: NOTA :: ISTO TA UM BOCADO OFF TOPIC, DEPOIS VER

$$M > \sum_{j=1}^{N} b_j$$

W por sua vez, verifica:

$$1 \leq W < M \text{ e } \gcd(W, M) = 1$$

O algoritmo considerado adopta uma estratégia brute force. Esta é possível devido à eficiência de calcular o gcd dos dois e no apresentado aqui [2]. Este considera um valor aleatório para W compreendido no intervalo apresentado acima, testando o gcd de este com M. Se forem coprimos o processo para, caso contrário somamos um a este e repetimos o teste. O algoritmo pode ser descrito como.

Algoritmo 2 Gerador de Coprimos - brute force

Recebe: M - um número

Devolve: W - um número coprimo com o argumento, $1 \leq W < M$

```
W ← número aleatório tal que 1 ≤ W < M
while gcd(M, W) ≠ 1 do
    W = W + 1
end while
```

A sua implementação segue - ref.

2.4 Geração da Chave

Dada uma permutação de uma sequência π.
Dada uma sequência super crescente $b_1, ..., b_n$.
Dados dois inteiros co primos W e M.
A geração da chave pode ser dividida em duas subcategorias.

2.4.1 Geração da chave Pública

Definição 2. São computados a_i tal que:

$$a_i = W \times b_{\pi_i} modM \text{ para cada } i \text{ tal que } 1 \leq i \leq n.$$

A chave publica é dada por $(a_1, a_2, ..., a_n)$

2.4.2 Geração da chave Privada

A chave privada é dada por $(\pi, M, W, (b_1, b_2, b_3, ..., b_n))$ [3]

2.4.3 Versão Multi-Iterada

Uma variante do algoritmo base de Merkle-Hellman envolve disfarçar a sequência super crescente, através de sucessivas multiplicações modulares.
Neste caso precisamos de fixar outro inteiro, T, que ditará o numero de iterações para geração da chave. Mais uma vez um inteiro N é fixo como parâmetro de sistema.

Para $1 \leq j \leq T$ são feitos os cálculos descritos na definição ja nao e definicao. Para calcular a sequencia atual são calculados M_j e W_j, da mesma maneira que eram calculados na geração da chaves no algoritmo básico. O M_j é calculado com base na sequência super crescente anterior (índice $j-1$)

sendo então depois calculado W_j como descrito na definição **??**, tendo como base uma sequencia $a_1, ..., a_n$ dada, sendo esta a chave privada.

2.5 Encriptacão

A encriptação é feita a partir da chave pública $a_1, a_2, ...a_n$. É representada a mensagem m a encriptar em binário como uma string de tamanho N. $(m = m_1 m_2 ... m_n)$
Finalmente é calculado um inteiro c tal que: $c = \sum_{i=1}^{N} a_i \times m_i$.

2.6 Decriptação

Para recuperar a mensagem m a partir de c, o dono da chave privada deve:
Calcular $d = W^{-1} \times c \bmod M$.
Encontrar inteiros $r_1, ..., r_n, r_i \in \{0, 1\}$ tal que $d = r_1 \times b_1 + r_2 \times b_2 + ... + r_n \times b_n$.
Os bits da mensagem são então $m_i = r_{\pi_i}, i = 1, 2, ...n$.

2.6.1 Versão Multi-Iterada

A decriptação desta variante do algoritmo é análoga à sua encriptação, ou seja, os cálculos efetuados são os mesmos do que na versão básica do algoritmo, mas são efetuados mais vezes.
A decriptação é feita a partir de sucessivamente calcularmos:

$$d_j = W^{-1} \times d_{j+1} \bmod M_j, \text{ para } j = t, t-1, ..., 1$$

onde $d_{t+1} = c$.
Finalmente só temos de encontrar inteiros $r_1, ..., r_n, r_i \in \{0, 1\}$ tal que:

$$d_1 = r_1 \times a_1 + r_2 \times a_2 + ... + r_n \times a_n.$$

E os bits da mensagem são recuperados a partir de aplicarmos a permutação π.

Capítulo 3

Conclusões

Após a realizaçao do trabalho e depois de compreendido o problema em questão foi possivel concluir que o sistema Merkle-Hellman Knapsack tal como praticamente todos os cripto-sistemas baseados no *knapsack problem* nao é seguro.Tal como ficou provado, apenas alguns anos após a sua publicaçao, por **Adi Shamir**. Existem várias maneiras de o atacar.È possivel faze-lo,por exemplo, atacando a sua versão Multi-Iterada ou através da divisão em várias partes (falta-me aqui a palavra apropriada) , existindo ainda mais alguns ataques possiveis. Sendo uma demonstraçao de um ataque a este sistema é uma boa aposta para um trabalho futuro. Para finalizar, resta só concluir que existem poucos sistemas que sejam eguros e poderosos que ainda não tenham sido quebrados. Um deles é o RSA fazendo por isso com que seja um dos, se nao mesmo o mais utilizado.

Bibliografia

[1] Wikipedia, *Fisher–Yates shuffle — Wikipedia, The Free Encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Fisher%E2%80%93Yates%20shuffle&oldid=882337194`, Acedido em 05-May-2019.

[2] *Finding a coprime of a general magnitude*, `https://math.stackexchange.com/questions/2430742/finding-a-coprime-of-a-general-magnitude`.

[3] A. J. Menezes, *Handbook of applied cryptography*. CRC Press, 1997, cap. 8.6, Acedido a partir de `http://cacr.uwaterloo.ca/hac/about/chap8.pdf`, ISBN: 978-0849385230.

Apêndice A

Código??

Será que vai num ficheiro separadamente?