Universidade do Minho - Dep.to Informática
2º Semestre, 2022/2023
Mestrado em Engenharia Informática
Gestão e Segurança de Redes

Ficha de Trabalho Prático Nº1

SNMPkeyShare

Versão 3.0

Objectivos:

- Consolidação dos conhecimentos sobre os protocolos, mecanismos e filosofias da arquitetura de gestão *Internet-standard Network Management Framework* (INMF), dando especial relevo aos aspetos de interação protocolar, segurança e controlo de acesso.

Observações:

- O trabalho deverá ser realizado ao longo de 30 a 40 horas efetivas de trabalho individual.

Requisitos:

- Acesso a sistema com, pelo menos, um pacote *freeware* instalado com suporte a SNMP (versão 2, no mínimo): **Net-SNMP**, CMU-SNMP, SCOTTY, etc.
- Utilização opcional de APIs de programação que facilitem a implementação de primitivas SNMPv2c ou SNMPv1.

AVISOS:

- Não serão tolerados atropelos aos direitos de autor de qualquer tipo de *software*...

Bibliografia específica e material de apoio

Material de apoio:

- Manuais/Tutoriais do *net-snmp*;
- MIBs em `/usr/share/snmp/mibs` (ou diretoria equivalente da instalação);
- Recurso `http://net-snmp.sourceforge.net/wiki/index.php/Tutorials/`;
- Recurso `http://www.simpleweb.org/`;
- Recurso `http://www.snmplinks.org/`.

Bibliografia:

- M. Rose, *The Simple Book*, Second Edition, Prentice Hall, 1996.
- B. Dias, *Gestão de Redes*, PAPCC, Universidade do Minho, 1996.
- W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Addison-Wesley, 2000.
- D. Mauro, K. Schmidt, *Essential SNMP*, O'Reilly, 2001.
- Ver outros recursos na área de "Conteúdo" no BB da UC e no material fornecido no início do semestre.

Modelo SNMPkeyShare

A inclusão de mecanismos de segurança para garantia de privacidade, autenticação e verificação da integridade dos dados foram definidos na segunda versão do INMF. No entanto, a sua implementação era opcional pelo que as duas primeiras versões do modelo, que passaram a ser conhecidas como SNMPv1 e SNMPv2c, não oferecem mecanismos de segurança nativos. Esses mecanismos são apenas garantidos pela terceira versão do INMF, comumente conhecida como SNMPv3, sendo que, na prática, não existe razão para os equipamentos e aplicações implementarem uma eventual versão SNMPv2. Ou implementam uma versão que não suporta mecanismos de segurança (v1 ou v2c) ou implementam uma versão que os suporta (v3).

Uma das limitações do modelo de segurança do SNMP é a falta de mecanismos normalizados para gestão e partilha de chaves (tanto de chaves para criptografia simétrica como para criptografia assimétrica). O principal objetivo deste trabalho é a definição e implementação dum sistema de gestão simplificado para criação e partilha de chaves para utilização em sistemas de gestão ou até em quaisquer sistemas distribuídos que necessitem dum sistema externo para gerir as suas chaves (ou segredos). Este sistema será designado de SNMPkeyShare.

O SNMPkeyShare será implementado sobre um modelo análogo à arquitetura SNMP, ou seja, o coração do sistema é um agente parecido a um agente SNMP que implementará a instrumentação duma MIB especial simplificada. A comunicação com o agente SNMPkeyShare será feita através duma versão simplificada do SNMP. Os gestores serão as aplicações que necessitam gerar e partilhar chaves entre si utilizando indiretamente o agente SNMPkeyShare.

As principais funcionalidades do agente SNMPkeyShare serão:

i. Ler um ficheiro de configuração F que deve ser lido no processo de arranque; este ficheiro define os parâmetros de funcionamento geral do agente (porta UDP de atendimento, etc.);
ii. Manter um objeto de gestão que indique (em segundos) há quanto tempo o agente iniciou/reiniciou a sua execução (*timespamp* S);
iii. Gerar uma matriz bidimensional Z (K colunas por K linhas) que irá servir como estrutura de dados base para a geração de chaves de K bits de comprimento a partir duma chave mestra M de 2xK bytes de comprimento;
iv. Processar e atualizar a matriz Z a cada intervalo de T milissegundos;
v. Atender um pedido de geração duma chave enviado por uma aplicação e devolver o seu valor C e o seu identificador D;
vi. Manter uma tabela de informação direta de todas chaves geradas (o seu identificador D, o seu valor C e um *timestamp* S_D) e a identificação E da entidade/aplicação que fez o pedido; as chaves só devem ter uma validade de V segundos, i.e., só devem ser guardadas informações na tabela durante V segundos; o número de entradas nesta tabela não deve exceder o valor máximo de X;
vii. Atender um pedido de verificação duma chave D enviado por uma aplicação (este pedido é indireto, i.e., qualquer verificação de informação mantida no agente tem de ser feita através da primitiva *get*); todos os pedidos têm um identificador próprio e unívoco P no contexto da aplicação/gestor que, para a aplicação que faz o pedido, deve ser considerado válido durante V segundos; depois de V segundos sem obter uma resposta, a aplicação pode partir do princípio que o agente não recebeu o pedido, ou que, tendo recebido, não respondeu, ou que, tendo respondido, a resposta não chegou ao destino.

Não existem quaisquer requisitos de segurança no sistema, i.e., não é necessário implementar quaisquer mecanismos de autenticação, confidencialidade e controlo de acesso. As únicas restrições têm a ver com a configuração inicial do funcionamento do sistema, ou seja, os valores dos parâmetros K, M, T, V e X devem ser definidos apenas na altura do arranque do agente através do ficheiro F.

A comunicação com o agente deve ser feita através dum protocolo com as mesmas características do SNMP, mas com um PDU simplificado, com menos primitivas, sem inclusão de mecanismos de segurança e encapsulado em UDP.

Geração e manutenção das chaves

O objetivo iii referido na secção anterior implica a geração duma matriz Z de K x K bytes a partir do valor da chave M que tem um comprimento de 2K bytes (ou caracteres da tabela de códigos ASCII estendido). Seja Z[i,j] o elemento na linha i e coluna j da matriz Z ($0 \leq i < K$ e $0 \leq j < K$), o algoritmo de geração desta matriz Z é o seguinte:

1. Seja M_1 uma sequência com os primeiros K bytes de M e M_2 outra sequência com os K bytes restantes de M (ou seja, $M = M_1+M_2$);
2. Criar uma matriz Z_A de K x K bytes em que a primeira linha é igual a M_1 e as restantes linhas são iguais a *rotate*(M_1,i), em que i é o índice da linha e a função *rotate*(M,n) retorna uma sequência com o mesmo comprimento de M mas com os seus elementos rodados pela direita n vezes; por exemplo, *rotate*("012345",1) = "501234";
3. Criar uma matriz Z_B de K x K bytes em que a primeira coluna é igual a *transpose*(M_2) e as restantes colunas são iguais a *transpose*(*rotate*(M_2,j)), em que j é o índice da coluna e a função *transpose*(M) retorna uma sequência com o mesmo comprimento de M mas com os seus elementos *alinhados/organizados verticalmente*, de *cima para baixo*, numa coluna, se M for uma sequência com os elementos *alinhados/organizados horizontalmente*; se M já for uma sequência com os elementos *alinhados/organizados verticalmente* a função retorna uma sequência com o mesmo comprimento de M mas com os seus elementos *alinhados/organizados horizontalmente*, da *esquerda para a direita*, numa linha;
4. Criar uma matriz Z_C de K x K bytes em que $Z_C[i,j]$ = *random*($Z_A[i,j]$,0,255), em que a função *random*(seed,min,max)[1] retorna um número aleatório entre min e max usando a semente seed; criar uma matriz Z_D de K x K bytes em que $Z_D[i,j]$ = *random*($Z_B[i,j]$,0,255); em alternativa pode ser criada apenas uma matriz Z_S de K x K bytes em que $Z_S[i,j]$ = *random*(S,0,255);
5. Calcular uma matriz Z de K x K bytes em que Z[i,j] = *xor*($Z_A[i,j]$,$Z_B[i,j]$,$Z_C[i,j]$,$Z_D[i,j]$) ou, em alternativa, Z[i,j] = *xor*($Z_A[i,j]$,$Z_B[i,j]$,$Z_S[i,j]$), e em que *xor*(A,B,C,…) devolve um byte que é o resultado de aplicar a função booleana *xor* (*Exclusive OR*) aos seus argumentos (bytes) duma forma binária (bit a bit); a função *xor* pode ser substituída por uma outra função *fm*(a,b) que devolve um valor a partir de dois argumentos de tal forma que o valor devolvido e os argumentos têm valores possíveis no mesmo limite e todos os valores possivelmente devolvidos são uniformemente distribuídos; a implementação desta de *fm* pode apresentar vantagens na implementação em relação à função *xor* se for implementada à custa dum simples acesso a um array predefinido e quando os valores possíveis dos argumentos são limitados (neste caso, a função *fm* pode ser implementada como o acesso direto a um elemento dum array predefinido no ficheiro F).

O objetivo iv referido na secção anterior implica processar e atualizar a matriz Z de K x K bytes, criada no passo anterior, com o resultado desse processamento a cada T milissegundos, conforme o algoritmo seguinte:

1. Seja Z_{i*} a linha i de Z, i.e., $Z_{i*}[j] = Z[i,j]$, atualizar a matriz Z de acordo com Z_{i*} = *rotate*(Z_{i*},*random*(Z[i,0],0,K-1));
2. Depois de aplicada a rotação do passo anterior, atualizar a matriz Z de acordo com Z_{*j} = *rotate_vertical*(Z_{*j},*random*(Z[0,j],0,K-1)), em que Z_{*j} é a coluna j de Z, i.e., $Z_{*j}[i] = Z[i,j]$, e em que *rotate_vertical*(M,n) roda verticalmente n vezes (de cima para baixo) a sequência M (função análoga a *rotate* mas para ser aplicada a colunas em vez de linhas duma matriz).

[1] Em todos os algoritmos, em alternativa a uma função de *random*(semente,min,max) que incluiu um argumento para definição da semente, pode usar-se a função mais simples *random**(min,max) sem definição duma semente.

O objetivo v referido na secção anterior implica a geração duma chave C após a receção dum pedido de chave por parte dum gestor/aplicação (este tipo de pedido tem de ser feito indiretamente através duma primitiva *set*). Sempre que um pedido destes é aceite e uma nova chave C é devolvida, o algoritmo anterior de atualização da matriz Z tem de ser executado antes de outros pedidos de geração de chave serem atendidos. Segue-se o algoritmo de geração duma chave C com K bytes de comprimento:

1. Seja N o número de vezes que a matriz já foi atualizada desde que o agente se iniciou;
2. Escolhe-se uma linha Z_{i*} de Z, de tal forma que i = *random*(N+Z[0,0],0,K-1);
3. Escolhe-se uma coluna Z_{*j} de Z, de tal forma que j = *random*(Z[i,0],0,K-1);
4. Calcula-se a chave através da expressão C = *xor*(Z_{i*},*transpose*(Z_{*j})); em alternativa à função *xor* pode ser aplicada a função *fm* definida anteriormente;

Comunicação SNMPkeyShare

O protocolo comunicacional do sistema é baseado no protocolo de comunicação SNMPv1. As alterações necessárias servem para tornar o protocolo ainda mais simples de implementar e são complementadas por uma definição duma base de dados de objetos de gestão especial: SNMPkeyShare MIB

Assim, na comunicação entre os gestores/aplicações e o agente podem ser usadas apenas três primitivas:

- snmpkeyshare-get(P,N_L,L) – Esta primitiva permite a uma aplicação/gestor obter, do agente que receber o pedido, os valores das N_L instâncias dos objetos de gestão indicadas na lista de identificadores de instâncias L ($N_L \geq 1$); a lista L consiste em N_L pares de valores (I-ID,N) que identificam uma instância referenciada por I-ID e mais N instâncias lexicograficamente seguintes à instância referenciada por I-ID ($N \geq 0$); a identificação do pedido (número inteiro) é unívoca no contexto da aplicação/gestor; não é permitido que durante V segundos o gestor identifique outro pedido com o mesmo P, sendo até aconselhável que o gestor não utilize valores para P repetidos num intervalo temporal muito maior que V segundos; não é obrigatório que o agente responda, nem que responda num intervalo de tempo qualquer, nem que responda aos pedidos numa ordem qualquer pré-definida;
- snmpkeyshare-set(P,N_W,W) – Esta primitiva permite a uma aplicação/gestor tentar modificar, no agente que receber o pedido, os valores das N instâncias dos objetos de gestão indicadas na lista W ($N_W \geq 1$); esta lista inclui N_W pares (I-ID,H) em que I-ID é um identificador duma instância e H é o valor desejado para essa instância; a identificação do pedido (número inteiro) é unívoca no contexto da aplicação/gestor; não é permitido que durante V segundos o gestor identifique outro pedido com o mesmo I-ID, sendo aconselhável que o gestor não utilize valores para os I-ID repetidos num intervalo temporal muito maior que V segundos; não é obrigatório que o agente obedeça ao pedido de *set*, nem que responda num intervalo de tempo qualquer, nem que responda aos pedidos numa ordem qualquer pré-definida, mas é obrigatório que o agente responda ao pedido se chegar a implementar o pedido de *set*;
- snmpkeyshare-response(P,N_W,W,N_R,R) – Esta primitiva permite a um agente responder aos pedidos *get* e *set* dos gestores/aplicações; o valor de P informa ao gestor a que pedido original a resposta se refere; a resposta inclui os valores das N_W instâncias dos objetos de gestão indicadas na lista W ($N_W \geq 1$); esta lista inclui N_W pares (I-ID,H) em que I-ID é um identificador duma instância e H é o valor dessa instância na implementação da MIB no agente; na lista W, o agente tem de referenciar todas as instâncias referenciadas pela lista L do pedido P original e que existam no agente e em que o seu valor possa ser devolvido pelo agente ao gestor que irá receber a resposta; no caso de a resposta ser a um pedido *set*, o agente deve indicar na lista W os valores com que as instâncias realmente ficaram depois do pedido *set* ter sido executado; no caso de não haver nenhum valor de instância válido a ser transmitido, $N_W = 1$, I-ID = 0 e H = 0; a lista R inclui todos os erros encontrados durante o processo de processamento do pedido P no agente; esta lista inclui N_R pares (I-ID,E) em que I-ID é um identificador duma instância e E é o valor que define qual o erro associado ao processamento do pedido P em relação a essa instância; se

não houver erros a reportar, a lista R incluirá apenas um elemento/par a indicar que não existiram erros no processamento do pedido no agente (neste caso, $N_R = 1$, I-ID = 0 e E = 0).

As três primitivas anteriores devem ser encapsuladas na mesma unidade protocolar de dados (PDU) que tem os seguintes nove campos:

1. Identificação do modelo de segurança (S) – Número inteiro que identificará quais os mecanismos de segurança a utilizar; se for igual a zero então não se aplicam mecanismos de segurança (neste trabalho o único valor de S válido é zero);
2. Número de parâmetros necessários à implementação dos mecanismos de segurança (N_S) – Se S for igual a zero então N_S também é obrigatoriamente igual a zero;
3. Lista dos parâmetros necessários à implementação dos mecanismos de segurança (Q) – O tipo de parâmetros depende do modelo de segurança; se S e N_S forem iguais a zero, então Q é uma lista vazia;
4. Identificação do pedido (P) – Número inteiro (ver explicação das primitivas);
5. Identificação do tipo de primitiva (Y) – Um número inteiro que identifica um dos três tipos de primitivas possíveis (0 = *response*, 1 = *get*, 2 = *set*);
6. Número de elementos da lista de instâncias e valores associados (N_L ou N_W) – Número inteiro que indica a quantidade de pares duma lista L (primitiva *get*) ou duma lista W (primitiva *set* ou primitiva *response*);
7. Lista de instâncias e valores associados (L ou W) – Lista L (primitiva *get*) ou lista W (primitiva *set* ou *response*); esta lista é formada por pares de valores (I-ID,H ou N) (ver explicação das primitivas);
8. Número de elementos da lista de erros (N_R) – Número inteiro que indica a quantidade de erros reportados na primitiva (ver explicação das primitivas) através da lista R;
9. Lista de erros e valores associados (R) – A lista R inclui todos os erros encontrados durante o processo de processamento do pedido P no agente (ver explicação das primitivas); no caso das primitivas *get* temos $N_R = 1$, I-ID = 0 e E = 0.

Todos os valores dos campos do PDU são codificados como *strings* ASCII (tabela estendida de códigos), i.e., sequências de caracteres terminadas com o valor NULL = 0.

MIB SNMPkeyShare

Esta MIB especial deve seguir um conjunto de regras mais simples que as definas na norma SMI para as MIB normal do SNMP e deve permitir implementar as funcionalidades anteriormente indicadas.

A MIB SNMPkeyShare deve apenas permitir o uso de três tipos simples de objetos/variáveis de gestão: strings (sequências de bytes ou caracteres terminados com o valor NULL), valores inteiros positivos e identificadores I-ID. Em termos de tipos compostos, esta MIB deve apenas permitir a definição de tabelas duma forma análoga às MIB do SNMP (mas o mais simplificada possível).

Nesta MIB especial, os objetos de gestão não precisam de estar organizados em grupos e a sua identificação faz-se através do tipo I-ID (tipo análogo aos OID do SNMP). Os valores do tipo I-ID não devem ter em consideração a identificação de objetos fora da própria MIB e os valores das chaves que identificam as instâncias só podem ser números inteiros positivos (o valor zero indica a primeira instância).

A definição correta da MIB SNMPkeyShare é um dos passos mais relevantes para o bom desenvolvimento do trabalho. É incluída, de seguida, uma versão simples, mas completamente funcional desta MIB.

```
-- groups in SNMPKEYSHARE
-- the OID of snmpKeyShareMib is irrelevant (it can be implemented as NULL)

system OBJECT IDENTIFIER ::= { snmpKeyShareMib 1 }
config OBJECT IDENTIFIER ::= { snmpKeyShareMib 2 }
keys OBJECT IDENTIFIER ::= { snmpKeyShareMib 3 }

-- the system group

systemRestartDate OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The date (YY*10^4+MM*10^2+DD) when the agent has started
a new Z matrix.”
::= { system 1 }

systemRestartTime OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The time (HH*10^4+MM*10^2+SS) when the agent has started
a new Z matrix.”
::= { system 2 }

systemKeySize OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The number of bytes (K) of each generated key.”
::= { system 3 }

systemIntervalUpdate OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The number of milliseconds of the updating interval of the
internal Z matrix.”
::= { system 4 }

systemMaxNumberOfKeys OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The maximum number of generated keys that are still valid.”
::= { system 5 }

systemKeysTimeToLive OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The number of seconds of the TTL of the generated keys.”
::= { system 6 }
```

```
-- the config group
-- the Z matrix and the fm(a,b) matrix function don’t need
-- to be defined in the MIB as they’re only needed for internal computations

configMasterKey OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The master double key M with at least K*2 bytes in size.”
::= { config 1 }

configFirstCharOfKeysAlphabet OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The ASCII code of the first character of the alphabet
used in the keys (default=33).”
::= { config 2 }

configCardinalityOfKeysAlphabet OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “The number of characters (Y) in the alphabet
used in the keys (default=94).”
::= { config 3 }

-- the data group
-- includes the table with the information from all
-- created keys that are still valid

dataNumberOfValidKeys OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The number of elements in the dataTableGeneratedKeys.”
::= { data 1 }

dataTableGeneratedKeys OBJECT-TYPE
SYNTAX SEQUENCE OF DataTableGeneratedKeysEntryType
ACCESS not-accessible
STATUS mandatory
DESCRIPTION "A table with information from all created keys that are still valid.”
::= { data 2 }

dataTableGeneratedKeysEntry OBJECT-TYPE
SYNTAX DataTableGeneratedKeysEntryType
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION “A row of the table with information for each key.”
INDEX { keyId } ::= { dataTableGeneratedKeys 1 }

DataTableGeneratedKeysEntryType ::=
SEQUENCE { keyId keyValue KeyRequester keyExpirationDate keyExpirationTime
keyVisibility}
```

```
keyId OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The identification of a generated key.”
::= { dataTableGeneratedKeysEntry 1 }

keyValue OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The value of a generated key (K bytes/characters long).”
::= { dataTableGeneratedKeysEntry 2 }

KeyRequester OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The identification of the manager/client that initially
requested the key.”
::= { dataTableGeneratedKeysEntry 3 }

keyExpirationDate OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The date (YY*10^4+MM*10^2+DD) when the key will expire.”
::= { dataTableGeneratedKeysEntry 4 }

keyExpirationTime OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-only
STATUS current
DESCRIPTION “The time (HH*10^4+MM*10^2+SS) when the key will expire.”
::= { dataTableGeneratedKeysEntry 5 }

keyVisibility OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION “0 – Key value is not visible; 1 – key value is only visible
to the requester; 2 – key value is visible to anyone.”
::= { dataTableGeneratedKeysEntry 6 }

-- when a manager/client wants to request a generation of a key it
-- sends a set() request to write 0,1 or 2 into the keyVisibility.0 instance;
-- the agent will create a new key (Id=KI) and will answer with the value
-- of the new keyVisibility.KI instance;
-- the manager/agent will then retrieve any required information from the agent,
-- including the keyValue.KI value, using get() requests.
```

Relatório e outras recomendações

O código dos protótipos do agente e do gestor podem ser desenvolvidas numa linguagem de programação à escolha. Podem usar-se APIs, funções ou excertos de código de terceiros para implementar aspetos tecnológicos, desde que devidamente referenciados. Em caso de dúvida consulte o docente para verificar a adequação da sua utilização no contexto do trabalho.

O código deve ser claro e usar convenções de nomeação de variáveis, tipos, funções e constantes. O código deve ser estruturado duma forma o mais modular possível, sem complexidades desnecessárias, e permitir reutilização sempre que possível. A qualidade e correção do código não se mede pelo seu tamanho! Os alunos devem documentar/explicar o código criado através de comentários nas secções mais relevantes e no relatório. Todos os ficheiros do código devem ter um cabeçalho com a informação relevante que identifique os seus autores e explique as principais funções/classes criadas, tipos de dados e variáveis usadas, etc.

O relatório pode ser escrito no formato e no editor que for mais conveniente e deve incluir, no mínimo:

- Uma primeira página com o título do trabalho e a identificação do autor (incluindo fotografia), universidade, curso, unidade curricular e data de entrega;
- Um índice do conteúdo;
- Uma secção com a discussão das estratégias escolhidas, as opções tomadas, os mecanismos e tecnologias adotados, incluindo eventuais otimizações;
- Uma secção com a definição e explicação detalhada da MIB, incluindo a forma de manipulação dos valores das instâncias dos objetos;
- Uma secção com a explicação e análise crítica das principais funções/classes implementadas, os seus principais méritos e a suas limitações mais importantes;
- Uma secção de conclusões que inclua uma eventual discussão sobre o que gostava de ter feito melhor ou de ter acrescentado e não conseguiu;
- Uma lista de eventuais referências bibliográficas, artigos científicos ou recursos informais na web e que tenham sido úteis.

O relatório é para ser avaliado pelo docente por isso não inclua informação genérica e irrelevante que o docente já conheça. Tente ser conciso e claro. Sempre que incluir uma afirmação importante no contexto do relatório e que seja de autoria de terceiros, ou que seja baseada diretamente em afirmações de terceiros ou concluída de informação retirada de recursos alheios, deve referenciar corretamente essas autorias ou proveniências e acrescenta-las na lista das referências.

O relatório pode incluir algumas partes relevantes do código quando estas ajudam às análises e justificações apresentadas. Não inclua código desnecessário no texto do relatório. Sempre que possível, comente antes o próprio código.

No material entregue inclua apenas dois ficheiros: um ficheiro PDF com o relatório e um ficheiro zip com todos os ficheiros do código do projeto dentro duma diretoria com o seguinte nome `GSR-22_23-Número_Aluno.zip`, como por exemplo, `GSR-22_23-83974.zip`.

Por fim, é recomendável que, durante a defesa do trabalho, tente responder honesta e concisamente apenas às questões colocadas por forma a que as sessões de apresentação não se arrastem muito para além dos 30-40 minutos.

Anexo: Exemplo

M= "07994506586870582927",
K=10

Gerar Z:

Za

0	7	9	9	4	5	0	6	5	8
8	0	7	9	9	4	5	0	6	5
5	8	0	7	9	9	4	5	0	6
6	5	8	0	7	9	9	4	5	0
0	6	5	8	0	7	9	9	4	5
5	0	6	5	8	0	7	9	9	4
4	5	0	6	5	8	0	7	9	9
9	4	5	0	6	5	8	0	7	9
9	9	4	5	0	6	5	8	0	7
7	9	9	4	5	0	6	5	8	0

Zb

6	7	2	9	2	8	5	0	7	8
8	6	7	2	9	2	8	5	0	7
7	8	6	7	2	9	2	8	5	0
0	7	8	6	7	2	9	2	8	5
5	0	7	8	6	7	2	9	2	8
8	5	0	7	8	6	7	2	9	2
2	8	5	0	7	8	6	7	2	9
9	2	8	5	0	7	8	6	7	2
2	9	2	8	5	0	7	8	6	7
7	2	9	2	8	5	0	7	8	6

Zs[i,j] = random*(0,9)

3	7	1	9	6	7	3	5	9	7
0	7	7	5	6	0	8	5	1	7
1	4	5	0	4	7	8	5	2	8
1	9	0	8	4	6	8	6	9	8
8	7	9	8	9	7	0	6	3	0
9	3	4	1	5	0	5	8	0	4
2	9	0	1	9	8	8	0	4	5
8	1	1	5	1	8	1	7	5	9
9	7	8	6	4	2	1	6	9	7
0	3	6	4	1	3	2	0	7	9

Z[i,j] = fm(fm(Za[i,j],Zb[i,j]),Zs[i,j])

5	1	1	9	9	6	9	8	9	9
6	7	1	3	2	7	3	0	6	8
6	0	5	3	4	1	2	3	0	1
6	9	6	9	5	5	0	9	8	1
1	8	9	9	5	1	6	2	0	6
8	9	8	6	3	6	8	3	3	8
3	8	0	6	9	9	9	3	4	8
0	2	1	0	6	1	7	7	8	8
8	1	2	5	2	1	6	5	5	1
3	9	2	8	1	9	1	8	9	5

fm

6	7	2	0	0	3	5	9	9	5	*0*
2	1	0	9	1	9	7	6	2	2	*1*
7	2	3	0	8	2	9	3	2	7	*2*
0	0	0	9	2	0	5	7	1	8	*3*
1	6	2	3	0	2	1	4	0	4	*4*
6	2	6	5	7	5	3	7	9	5	*5*
7	8	4	6	1	9	3	4	5	2	*6*
8	6	1	4	8	8	7	8	1	9	*7*
3	1	7	4	5	8	2	1	0	9	*8*
6	1	9	9	4	3	5	6	3	8	*9*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	

Atualizar Z:

Z

7	1	2	9	5	7	6	8	1	9
0	6	6	2	2	3	9	5	4	6
1	7	8	5	2	9	1	3	1	5
1	9	0	5	5	5	5	2	0	8
5	2	9	5	3	2	0	1	7	3
0	6	4	2	4	3	5	1	3	7
1	6	1	8	3	2	5	9	6	1
2	5	8	9	8	8	5	7	8	8
9	7	0	8	4	9	5	7	9	9
4	5	7	2	0	5	4	5	2	8

Z'i* =rotate(Zi*,random*(0,9)) *↓random*(0,9)*

6	8	1	9	7	1	2	9	5	7	*4*
6	0	6	6	2	2	3	9	5	4	*1*
5	2	9	1	3	1	5	1	7	8	*7*
2	0	8	1	9	0	5	5	5	5	*3*
1	7	3	5	2	9	5	3	2	0	*3*
2	4	3	5	1	3	7	0	6	4	*7*
9	6	1	1	6	1	8	3	2	5	*3*
8	2	5	8	9	8	8	5	7	8	*1*
9	7	0	8	4	9	5	7	9	9	*0*
7	2	0	5	4	5	2	8	4	5	*8*

Z*j = rotate_vertical(Z'i*,random*(0,9))

9	8	0	5	1	8	2	3	7	4
7	0	1	5	6	9	2	5	9	8
6	2	6	1	9	5	3	7	4	5
6	0	9	8	4	1	5	8	5	0
5	7	8	8	4	2	5	9	5	4
2	4	3	5	7	1	5	9	7	5
1	6	3	9	2	0	7	1	5	8
2	2	1	6	3	9	8	5	2	9
9	7	5	1	9	3	8	3	6	4
8	2	0	1	2	1	5	0	2	7

2 0 1 6 5 3 1 4 3 9 ←random(0,9)*

Calcular uma chave C a partir de Z: *i = random*(0,9) = 8, j = random*(0,9) = 3*

Zi*

7	1	2	9	5	7	6	8	1	9
0	6	6	2	2	3	9	5	4	6
1	7	8	5	2	9	1	3	1	5
1	9	0	5	5	5	5	2	0	8
5	2	9	5	3	2	0	1	7	3
0	6	4	2	4	3	5	1	3	7
1	6	1	8	3	2	5	9	6	1
2	5	8	9	8	8	5	7	8	8
9	7	0	8	4	9	5	7	9	9
4	5	7	2	0	5	4	5	2	8

i=8

Z*j

7	1	2	9	5	7	6	8	1	9
0	6	6	2	2	3	9	5	4	6
1	7	8	5	2	9	1	3	1	5
1	9	0	5	5	5	5	2	0	8
5	2	9	5	3	2	0	1	7	3
0	6	4	2	4	3	5	1	3	7
1	6	1	8	3	2	5	9	6	1
2	5	8	9	8	8	5	7	8	8
9	7	0	8	4	9	5	7	9	9
4	5	7	2	0	5	4	5	2	8

j=3

m1 = Zi*

9	7	0	8	4	9	5	7	9	9

m2 = transpose(Z*j)

9	2	5	5	5	2	8	9	8	2

fm

6	7	2	0	0	3	5	9	9	5	*0*
2	1	0	9	1	9	7	6	2	2	*1*
7	2	3	0	8	2	9	3	2	7	*2*
0	0	0	9	2	0	5	7	1	8	*3*
1	6	2	3	0	2	1	4	0	4	*4*
6	2	6	5	7	5	3	7	9	5	*5*
7	8	4	6	1	9	3	4	5	2	*6*
8	6	1	4	8	8	7	8	1	9	*7*
3	1	7	4	5	8	2	1	0	9	*8*
6	1	9	9	4	3	5	6	3	8	*9*
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	

C[i] = fm(m1[i],m2[i])

8	1	3	8	2	9	9	6	3	9