FIAP ON

CYBERSECURITY

CRIPTOGRAFIA:
FUNDAMENTOS E APLICAÇÕES

OSMANY D.R. DE ARRUDA

10

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

10 CRIPTOGRAFIA: FUNDAMENTOS E APLICAÇÕES

10.1 Visão geral

Criptografia é uma das técnicas mais eficazes e difundidas para garantia da confidencialidade por meio da cifragem da mensagem, tornando-a, inicialmente, ilegível. Assim sendo, entende-se como:

- Criptografar (também muitas vezes referenciado como encriptar), o processo de submissão da mensagem em clear text a um algoritmo específico que a codifica, tornando-a incompreensível.
- Decriptografar (também muitas vezes referenciado como desencriptar), a reversão do processo de criptográfico, restaurando a legibilidade e compreensão da mensagem original.
- Clear text o texto claro, explícito, com o qual a mensagem foi originalmente cunhada.
- Texto ou mensagem cifrada, como o resultado produzido pela submissão da mensagem ao algoritmo criptográfico.

Além da confidencialidade, a criptografia pode ser aplicada também a outros contextos, por exemplo, para garantia da autenticidade da mensagem, para verificação da integridade e não repúdio da mensagem, dentre outras possibilidades.

Pode-se citar como exemplos práticos da aplicação da criptografia, de forma transparente para o usuário a substituição do protocolo FTP pelo SFTP, a substituição do protocolo TELNET pelo SSH e a substituição do POP3 pelo POP3S. Em cada um dos três exemplos, os primeiros protocolos referidos (FTP, TELNET, POP3) trafegam suas mensagens em *clear text*, portanto, incapaz de garantir a confidencialidade da mensagem; enquanto as alternativas apontadas (SFTP, SSH e POP3S) utilizam-se de diferentes algoritmos criptográficos para garantir a confidencialidade da mensagem.

É importante que os algoritmos criptográficos sejam eficientes em relação ao uso dos recursos computacionais, como a capacidade de processamento do

sistema, a fim de minimizarem o *overhead* decorrente dos processos de criptografia e decriptografia das mensagens.

Os sistemas criptográficos podem ser implementados de duas maneiras diferentes: via *software*, opção geralmente mais barata, entretanto, lenta se comparada à implementação baseada em *hardware*. Isso decorre da maior demanda de recursos computacionais, característica da criptografia por *software*, uma vez que a criptografia por *hardware* é feita por circuitos especializados, geralmente ASICs (*Application Specific Integrated Circuit*), circuitos integrados especializados que tratam o processo criptográfico de forma autônoma e independente da CPU do sistema, tornando-o assim mais caro, porém, mais rápido e com menos *overhead* para o sistema do que a implementação por *software*.

A FORTINET é um reconhecido *player* de mercado que costuma utilizar a criptografia por *hardware* em seus *appliances* para criação de VPNs.

Para que um algoritmo criptográfico possa ser considerado seguro, a descoberta da chave utilizada no processo de criptografia deve ser praticamente impossível de ser descoberta, assim como a decriptografia do texto cifrado, sem a chave que o criptografou.

Dois sistemas de chaves poderão são utilizados: sistema de chave privada (ou chave secreta) ou sistema de chave pública. Adicionalmente, existe ainda o *message-digest*, mais conhecido como função *hash* (e a partir daqui referenciado simplesmente como *hash*), o qual não tem como objetivo a cifragem da mensagem, mas, sim, a verificação de sua integridade ou a criação de sua assinatura digital. As funções *hash* são caracterizadas por recebem uma entrada de comprimento arbitrário, produzindo um *string* de saída de comprimento fixo e curto (por exemplo, 128 ou 160 bits).

Para que as funções *hash* sejam consideradas funcionais, elas devem apresentar duas propriedades:

- Baixo nível de colisão, o que significa que a probabilidade de duas entradas distintas produzirem o mesmo *hash* na saída deverá ser praticamente nula, mesmo que as entradas sejam muito semelhantes.
- Garantir que será praticamente impossível reconstituir a mensagem de entrada a partir do *hash* de saída.

10.2 Sistemas criptográficos

Dois sistemas de chaves poderão são utilizados pelos algoritmos criptográficos: sistema de chave privada (ou chave secreta) ou sistema de chave pública.

10.2.1 Sistema de chave privada (criptografia simétrica)

Sistema simples, utilizado para criptografia simétrica, ou seja, uma única chave criptografa e decriptografa a mensagem, sendo conhecida por todos os integrantes da comunicação.

Dada sua simplicidade, esses algoritmos são bastante rápidos, entretanto, dessa simplicidade surgem, ao menos, dois sérios problemas: o primeiro relacionado à forma como a chave criptográfica será distribuída entre os interlocutores, o que exige um canal seguro, uma vez que qualquer um que venha a possuir a chave, poderá decifrar a mensagem. E o segundo problema: a impossibilidade de garantir o não repúdio da mensagem. Simplificadamente, não repúdio significa impedir que um indivíduo ou entidade venha a negar a execução de alguma ação particular relacionada a uma mensagem ou transação, ou ainda, questionar a validade delas. A figura a seguir representa a criptografia simétrica.

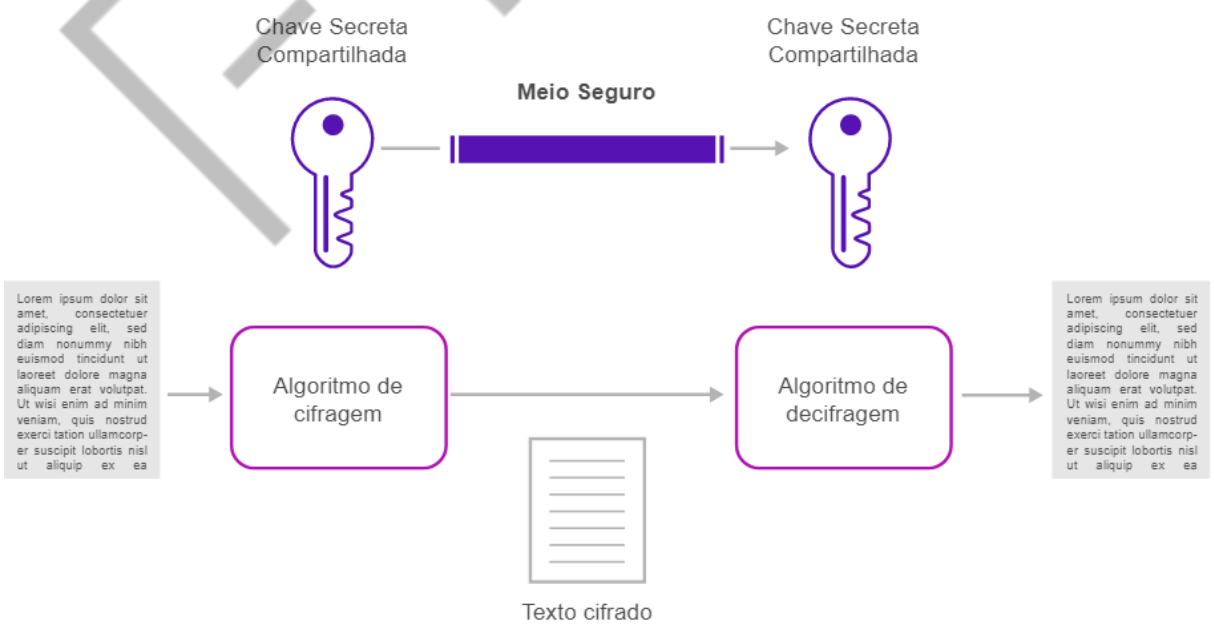

Figura 10.1 – Criptografia simétrica (chave privada)
Fonte: gta.ufrj.br (2020)

10.2.2 Sistema de chave pública (criptografia assimétrica)

Criado em 1976 por Whitfield Diffie e Martin Hellman, este sistema utiliza duas chaves distintas, uma **pública** e outra **privada**. Embora notoriamente mais seguro que a criptografia simétrica e tendo ainda resolvido o problema de gerenciamento das chaves, é consideravelmente mais complexo e, portanto, mais lento que ela. A criptografia assimétrica constitui excelente mecanismo para garantir a confidencialidade e a autenticidade da mensagem. Considere-se o cenário ilustrado na figura abaixo: Beto é o destinatário de uma mensagem cifrada com sua chave pública.

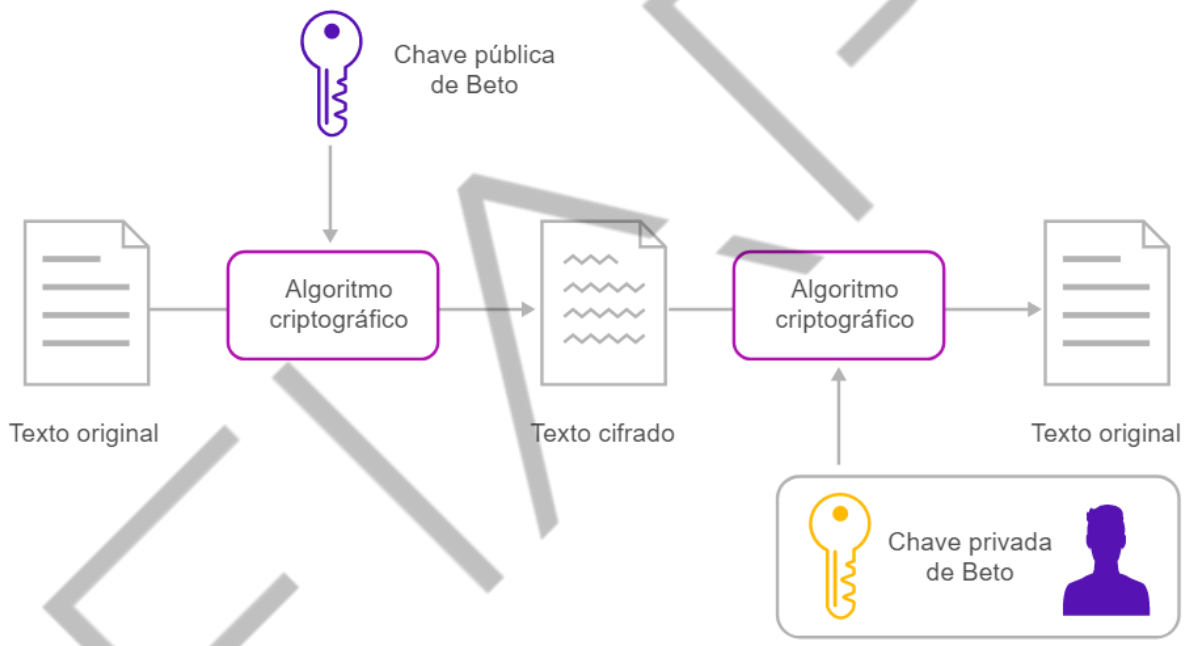

Figura 10.2 – Garantia da CONFIDENCIALIDADE
Fonte: Cartilha de Certificação Digital (2020)

Por meio de sua chave privada, que é secreta e não deve ser compartilhada com ninguém, somente Beto – destinatário da mensagem – será capaz de decifrá-la sendo assim assegurada a CONFIDENCIALIDADE de seu conteúdo. As chaves públicas podem ser compartilhadas por diversos meios diferentes, por exemplo, diretórios públicos disponíveis na Internet.

Qualquer mensagem cifrada com uma chave pública só poderá ser decriptografada por meio da chave privada correspondente.

Tome-se agora como referência um novo cenário: Alice é a autora de uma mensagem cifrada com sua chave privada e a seguir enviada a um ou mais destinatários.

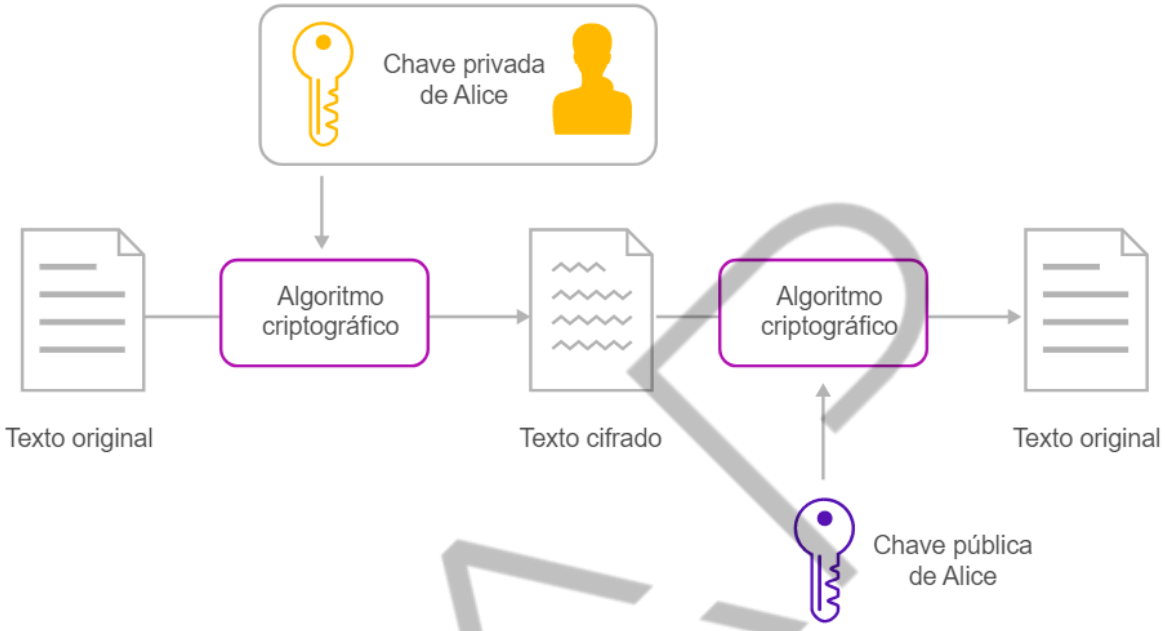

Figura 10.3 – Garantia da AUTENTICIDADE
Fonte: Cartilha de Certificação Digital (2020)

Tendo cifrado a mensagem com sua chave privada, Alice garante a AUTENTICIDADE dela, ou seja, a autoria de um documento ou a identificação em uma transação. Pode-se concluir, então, que a CONFIDENCIALIDADE e a AUTENTICIDADE utilizam os mesmos pares de chaves (pública e privada), porém, em sentidos inversos.

O destinatário de uma mensagem cifrada com uma chave privada só poderá decriptografá-la por meio da chave pública correspondente.

A criptografia assimétrica é baseada em funções matemáticas unidirecionais que impedem a recuperação da mensagem de entrada com base no resultado obtido na saída (ou ao menos, dificulta tanto tal procedimento que o torna computacionalmente inviável).

10.3 Algoritmos criptográficos

Existem vários algoritmos criptográficos simétricos e assimétricos, desenvolvidos para aplicação aos mais variados contextos.

10.3.1 Algoritmos criptográficos simétricos

10.3.1.1 DES e 3DES (Data Encryption Standard)

Algoritmo desenvolvido nos anos 1970 pelo NIST (*National Institute ou Standards and Technologies*) em conjunto com a IBM. O DES é uma cifra de bloco, ou seja, um bloco de entrada em texto claro – no caso com tamanho de 64 *bits*, é tratado como um todo a fim de produzir na saída, um bloco cifrado de mesmo tamanho. Entretanto, como a chave criptográfica do DES é comprimida, ela passa a ter tamanho de 56 bits sendo, portanto, considerado inseguro pelos padrões atuais.

No 3DES, o algoritmo DES é sequencialmente aplicado três vezes, com três chaves distintas de 56 *bits,* o que produz uma chave final com comprimento efetivo de 168 *bits*. No 3DES, a mensagem é cifrada pela primeira chave (K1), o resultado é decifrado pela segunda chave (K2), sendo então novamente cifrado pela terceira chave (K3), conforme ilustrado pela figura abaixo.

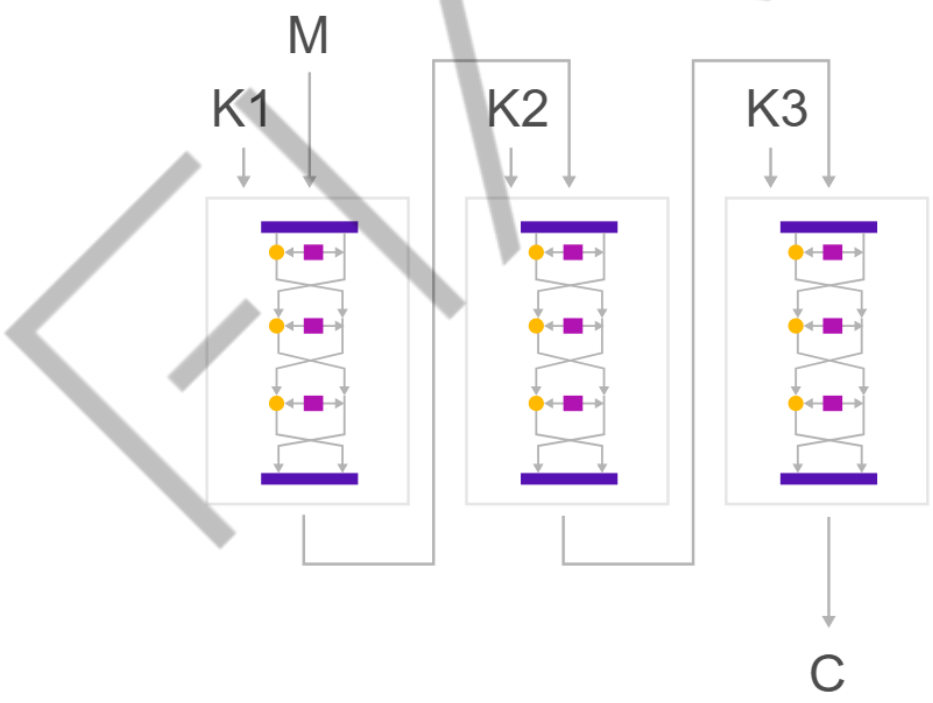

Figura 10.4 – Algoritmo 3DES
Fonte: Wikipedia.org (2018)

É possível ainda, em vez de utilizar-se de três chaves, a execução do processo utiliza apenas duas quando, então, a mensagem inicial é cifrada com a primeira chave, o resultado é decifrado pela segunda e novamente cifrado com a primeira, tendo-se assim, uma chave com tamanho efetivo de 112 *bits*.

10.3.1.2 RC2, RC4 e RC5

A exemplo do DES, o RC2 também é uma **cifra de blocos** que opera com entradas com 64 *bits,* podendo, entretanto, trabalhar com chaves criptográficas com até 2048 *bits*, o que pode torná-lo bem mais seguro que o DES.

O tamanho mais usual para a chave criptográfica é de 128 *bits*, o qual já ainda pode ser considerado como relativamente adequado. O RC2 pode ser um bom substituto para o DES, sendo em *softwares*, aproximadamente duas vezes mais rápido que ele.

Diferentemente dos dois anteriores, o RC4 é uma **cifra de fluxo**, o que significa que o fluxo de dados em texto claro é cifrado de forma contínua (*bit* a *bit*, *byte* a *byte* ou outras unidades de dados) com a ajuda de uma chave inserida por um gerador pseudo-aleatório. Como o RC2, o RC4 também pode trabalhar com chaves de tamanho variável (máximo 2048 *bits*, tipicamente 128 *bits*) podendo, em *softwares*, vir a ser até dez vezes mais rápido que o DES.

O RC5 é também uma técnica de cifragem em bloco, conhecido por sua flexibilidade e possibilidade de parametrização. No RC5 os blocos de entrada podem ter qualquer tamanho predeterminado (32, 64 ou 128 *bits*), o usuário pode definir o tamanho da chave (0-255 *Bytes*), sendo possível ainda, predeterminar o número de iterações do algoritmo (0-255).

10.3.1.3 Idea (International Data Encryption Algorithm)

O IDEA também é uma técnica de cifragem simétrica em bloco, a qual utiliza-se de blocos fixos com 64 *bits* e chaves com tamanho **fixo** de 128 *bits*, fato que poderá se tornar desfavorável no futuro.

10.3.1.4 Blowfish

Algoritmo conhecido por sua velocidade, sendo mais rápido que o RC2 e o IDEA. Também trabalha com blocos fixos de 64 *bits*, mas as chaves podem ter qualquer comprimento, tipicamente, 128 *bits*.

10.3.1.5 AES

O AES nasceu deu uma competição promovida pelo NIST para criação de um novo algoritmo que substituísse o DES, sendo o processo seletivo iniciado em 1997 e finalizado em 2000 com a vitória do algoritmo Rijndael, escrito por Vincent Rijmen e Joan Daemen.

O AES também é um algoritmo de bloco, cujo tamanho é fixo em 128 *bits* podendo, entretanto, o tamanho da chave criptográfica variar entre 128, 192 e 256 *bits*. Vale destacar que o algoritmo Rijndael e o AES não são exatamente iguais, uma vez que no primeiro, também os tamanhos de bloco podem ser especificados variando de 32 em 32 *bits* com tamanho mínimo de 128 *bits* e máximo de 256 *bits*. A tabela abaixo resume as principais características dos algoritmos referidos.

Tabela 10.1 – Principais características dos algoritmos criptográficos

Algoritmo	Tipo	Tamanho da chave
DES	Por bloco	56
IDEA		128
RC2 , RC5		1 a 2048
3DES		112 ou 168
AES		128, 192 ou 256
BLOWFISH		Tipicamente 128
RC5	Por fluxo	Definido pelo Usuário

Fonte: Elaborada pelo autor (2020)

10.3.2 Algoritmos criptográficos assimétricos

Diferentemente dos algoritmos simétricos, os assimétricos – também conhecidos como algoritmos de chave pública –, conseguem lidar de forma adequada com o gerenciamento das chaves, as quais passam agora, a ser duas: uma pública e outra privada.

10.3.2.1 Algoritmo de diffie-hellman

Este algoritmo tem como objetivo garantir que dois interlocutores possam trocar, de maneira segura, a chave (simétrica) a ser utilizada para cifragem das mensagens trocadas entre si.

A figura a seguir ilustra o processo de troca de chaves.

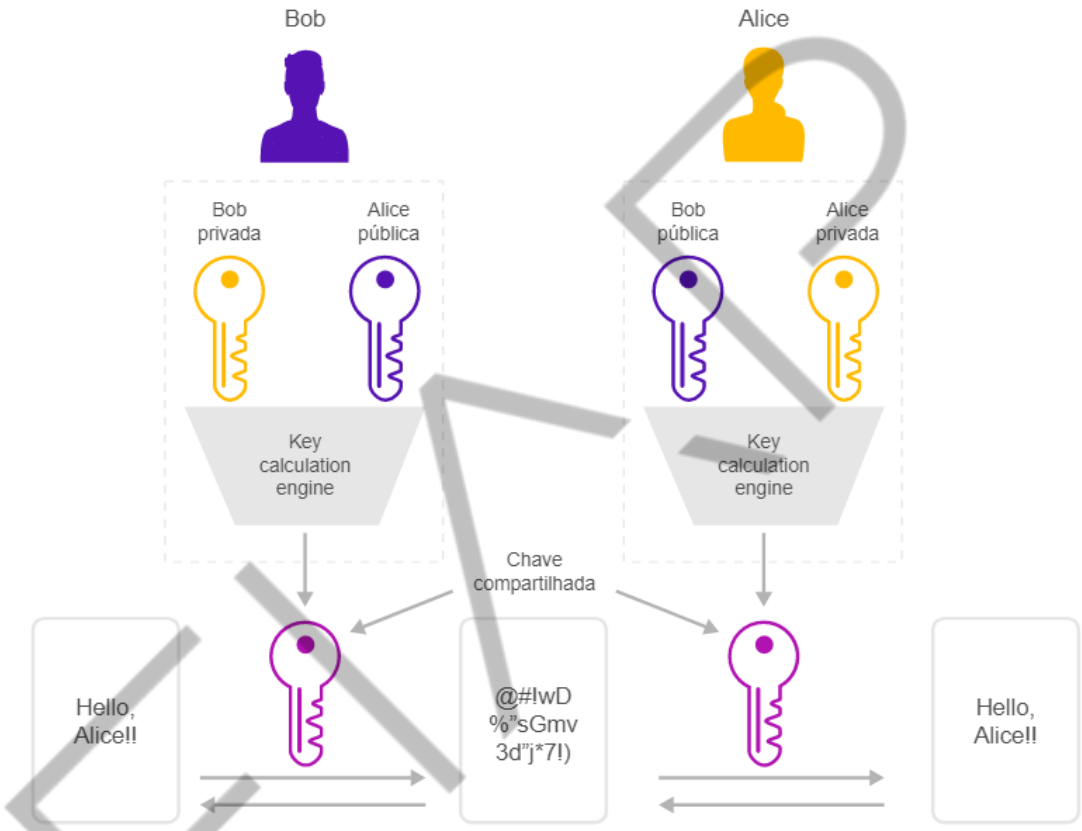

Figura 10.5 – Algoritmo de Diffie-Hellman
Fonte: Adaptada pelo autor (2020)

Na figura, Bob recebe cópia da chave pública de Alice, à qual é combinada sua chave privada, gerando, assim, uma chave compartilhada. Da mesma forma, Alice recebe uma cópia da chave pública de Bob e a combina à sua chave privada, obtendo assim, a mesma chave compartilhada. É importante salientar que no algoritmo de Diffie-Hellman, a combinação da chave privada A e da chave pública B gera o mesmo resultado que a combinação da chave privada B e da chave pública A, e ainda, que ele pode ser usado para distribuição de chaves, mas não pode ser usado para criptografar/decriptografar as mensagens.

10.3.2.2 RSA

Algoritmo inventado por Ron Rivest, Adi Shamir e Leonard Adleman, o RSA tem sua segurança diretamente relacionada à dificuldade de realizar fatorações, utilizando-se de grandes números primos (acima de 100 dígitos). O RSA pode assegurar a confidencialidade e a autenticidade da mensagem, entretanto, é considerado um algoritmo lento, inadequado à cifragem de grandes blocos de dados.

10.3.2.3 SSL / TLS

O *Secure Socket Layer* (SSL), desenvolvido em 1994 pela Netscape e RSA, estabelece um canal criptografado entre o servidor *web* e o navegador do cliente, a fim de garantir a PRIVACIDADE, INTEGRIDADE e AUTENTICIDADE dos dados trafegados.

Na primeira fase da comunicação, cliente e servidor verificam quais os algoritmos suportados por ambos que serão utilizados na comunicação (RSA, DAS, ECDSA). O cliente requisita a um servidor que suporte o protocolo SSL uma conexão segura, prontamente enviando uma lista com os algoritmos disponíveis para a cifragem e autenticação dos dados.

Na segunda fase, o servidor recebe a requisição e escolhe o mais seguro dentre os algoritmos listados pelo cliente, que ele também possua, e o avisa dessa decisão. Ambos trocam chaves e realizam a autenticação – são utilizados algoritmos de chave pública (RSA e Diffie-Hellman, dentre outros). Para se autenticar, o servidor envia sua identificação na forma de um certificado digital, o qual geralmente contém o nome do servidor, a Autoridade Certificadora para verificação e sua chave pública.

Na terceira fase, as mensagens são autenticadas por códigos gerados por funções *hash* HMAC-MD5 ou HMAC-SHA, garantindo, a partir daí, que as mensagens sejam trocadas com segurança após passarem pelo SSL Record Protocol. Geralmente, a autenticação do servidor é feita por intermédio de uma Autoridade Certificadora (AC). Nesse caso, o cliente se vale de uma chave pública da própria AC para validar sua assinatura no *site* do servidor. A AC deve estar na

lista de ACs confiáveis para ter certeza de que o servidor é quem ele diz ser. O TLS é o nome adotado pela IETF desenvolvimento de um protocolo de segurança padronizado baseado no SSL 3.0.

Há algumas pequenas diferenças entre o SSL e o TLS, entretanto, o protocolo permanece substancialmente o mesmo – tanto que o protocolo TLS 1.0 é por vezes identificado como SSL 3.1. É comum encontrar aplicações que suportam ambos (SSL/TLS), não sendo, todavia, os dois protocolos interoperáveis, ou seja, somente um deles deve ser escolhido no momento da negociação. O TLS vem substituindo o SSL, especialmente em projetos envolvendo servidores *open source*, já entre os clientes, o SSL3 costuma ser mais difundido em detrimento do TLS.

10.3.2.4 DSA

O algoritmo de assinatura digital (DSA - *Digital Signature Algorithm*) foi desenvolvido pelo governo norte-americano para assinaturas digitais. O DSA pode ser usado exclusivamente para assinar dados e não para cifrá-los. O processo de assinatura do DSA recai sobre uma série de cálculos baseados em um número primo selecionado.

Embora projetado para usar chaves com tamanho máximo de 1024 *bits*, chaves mais longas são agora suportadas. Quando o DSA é utilizado, o processo de criação da assinatura digital é mais rápido do que sua validação. Quando o RSA é utilizado, o processo de validação da assinatura digital é mais rápido do que a criação.

10.4 Algoritmos de Hashing

Conforme já anteriormente referido, estes algoritmos são utilizados para verificar a integridade da mensagem, certificando-se que mudanças imprevistas não tenham acontecido. A tabela a seguir mostra as principais características dos algoritmos de *hashing* considerados seguros pelo *Federal Information Processing Standards* (FIPS).

Tabela 10.2 – Propriedades dos algoritmos de *hash* seguros

Algoritmo	Tamanho da mensagem(*bits*)	Tamanho do bloco(*bits*)	Tamanho da palavra(*bits*)	Tamanho domessage-digest(*bits*)
SHA-1	$<2^{64}$	512	32	160
SHA-224	$<2^{64}$	512	32	224
SHA-256	$<2^{64}$	512	32	256
SHA-384	$<2^{128}$	1024	64	384
SHA-512	$<2^{128}$	1024	64	512
SHA-512/224	$<2^{128}$	1024	64	224
SHA-512/256	$<2^{128}$	1024	64	256

Fonte: Adaptada do FIPS (2020)

Embora o md5 não figure na lista, ele ainda é um dos algoritmos de *hash* mais utilizados, com *message-digest* de 128 *bits*. A figura abaixo ilustra a extração dos *hashes* md5, sha1 e sha256 do arquivo /var/log/messages no Linux.

Figura 10.6 – *Hashes* do arquivo /var/log/messages
Fonte: Elaborada pelo autor (2020)

Da figura pode-se observar que o *hash* (*message-digest*) md5 do arquivo tem /var/log/messages é composto por 32 dígitos hexadecimais, totalizando assim 128 *bits* (32x4=128). Da mesma forma, o *hash* sha1 do mesmo arquivo é composto por 40 dígitos hexadecimais, totalizando assim 160 *bits* (40x4=160) conforme apontado na tabela “Propriedades dos algoritmos de *hash* seguros”.

Digamos que alguém tenha feito uma cópia de um arquivo sigiloso, porém, com nome diferente do arquivo original a fim de mantê-la incógnita. Ao se extrair e comparar os *hashes* de ambos os arquivos, será possível atestar inequivocamente se tratar do mesmo arquivo sem, entretanto, abrir nenhum dos dois, já que o conteúdo é sigiloso. Ao se atribuir uma senha ao usuário, o *hash* é extraído e

armazenado no sistema, em vez da própria senha. Isso evita que sua senha venha a ser facilmente revelada, caso o arquivo onde se encontra armazenado venha a vazar.

Há vários *sites* na Internet especializados em "*crackear*" *hashes* para recuperação das respectivas senhas, tais como o HashKiller (https://hashkiller.co.uk/) e o CrackStation (https://crackstation.net/) dentre outros, bem como, softwares com a mesma finalidade, a exemplo do HashCat e John the Ripper.

O Hash_ID é um pequeno *script* em *python,* escrito por Zion3R, licenciado sob o modelo GPL3, capaz de identificar os mais diversos tipos de *hashes*, podendo servir como auxiliar em ataques por *brute force*. Para fins de teste, foi aberto um terminal, como usuário desprivilegiado, na máquina virtual Debian01 já anteriormente utilizada, e digitado o comando:

```
user1@debian01:~$ wget https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/hash-identifier/Hash_ID_v1.1.py
```

Esse comando faz o *download* do Hash_ID, valendo ressaltar aqui que, sendo ele um *script* em *python*, poderá ser executado em qualquer plataforma que tenha essa linguagem instalada. A figura a seguir ilustra o *download* da ferramenta.

Figura 10.7 – *Hashes* do Hash_ID
Fonte: Elaborada pelo autor (2020)

Para se observar o funcionamento do Hash_ID, serão gerados os *hashes* MD5 e SHA1 da *string* de referência Cyb3r-S3cur1ty. No Linux a extração desses *hashes* é muito simples, e são utilizadas duas ferramentas nativas: md5sum e sha1sum. No terminal aberto na VM Debian01, basta proceder conforme ilustrado pela figura que segue.

```
user1@debian01:~$ echo Cyb3r-S3cur1ty | md5sum
64d5fa200080a48d7eb7a908bf909e77  -
user1@debian01:~$ echo Cyb3r-S3cur1ty | sha1sum
1c75b063e0887beb209a894f5d78462ceac41917  -
user1@debian01:~$
```

Figura 10.8 – Extração do *hash* da *string*
Fonte: Elaborada pelo autor (2018)

No primeiro *prompt* da figura acima, observa-se a linha:

```
user1@debian01:~$ echo Cyb3r-S3cur1ty | md5sum
```

O comando *echo* imprime na tela o argumento a ele passado, no caso, a *string* Cyb3r-S3cur1ty. Entretanto, a intenção não é que esta *string* seja impressa na tela, mas sim, que seja passada ao comando md5sum, para que seja extraído seu *hash* MD5 (como se fosse uma senha digitada pelo usuário e armazenada pelo sistema), o que é feito por intermédio do *pipe* (|).

Analogamente, procede-se em relação à extração do *hash* SHA1, obtendo-se os resultados observados na figura. Para verificar se o Hash_ID consegue identificar o algoritmo utilizado, por meio do *hash* gerado, basta executá-lo, por meio da linha:

```
user1@debian01:~$ python Hash_ID_v1.1.py
```

Ao surgir a interface do Hash_ID, fornecer o *hash* a ser verificado conforme ilustrado pela figura “Extração do *hash* da *string*”, e o *script* o analisa de imediato, trazendo uma lista dos algoritmos com maior probabilidade de haver gerado o *hash*.

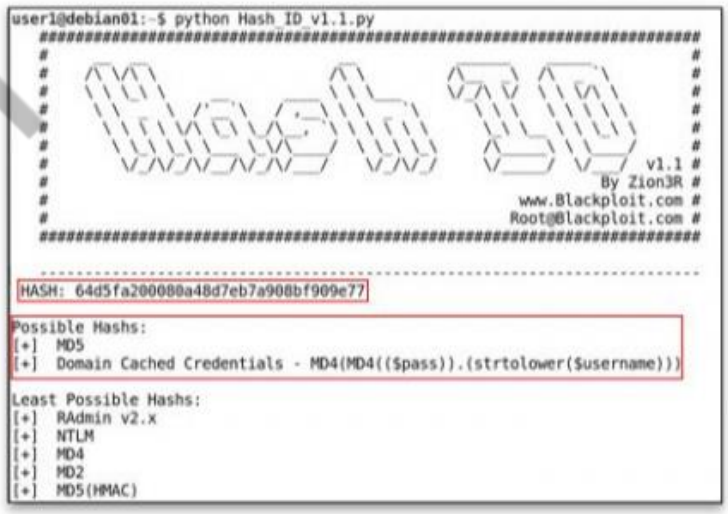

Figura 10.9 – Verificação do *hash* MD5
Fonte: Elaborada pelo autor (2020)

Pela figura acima pode ser observado que fornecendo ao Hash_ID o *hash* MD5 extraído da *string* de referência, o *script* imediatamente o reconhece como tal. Entretanto, embora o resultado esteja correto, sempre há um certo grau de incerteza no resultado assim, o que faz com que o Hash_ID elenque outros algoritmos com boa probabilidade de haverem gerado o *hash* testado.

A figura abaixo ilustra o teste feito com o *hash* SHA1 da string de referência.

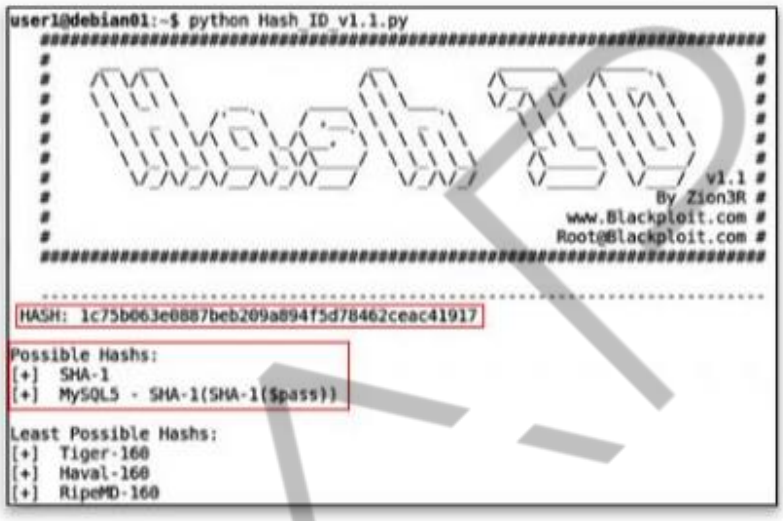

Figura 10.10 – Verificação do *hash* SHA1
Fonte: Elaborada pelo autor (2020)

10.5 tipos de cifras

Embora a criptografia tenha evoluído muito desde o surgimento dos primeiros algoritmos criptográficos, eles ainda servem como base conceitual para os algoritmos modernos.

10.5.1 cifras de substituição

As cifras de substituição são as primeiras cifras conhecidas. Simplesmente substituíam os caracteres do texto claro (individualmente ou em grupos) por outros caracteres, números ou símbolos. Tome-se como exemplo a substituição de cada letra do alfabeto por um número, como mostra a figura a seguir.

Tabela de substituição																									
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	W	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Mensagem em texto claro:								C	R	I	P	T	O	G	R	A	F	I	A						
Texto cifrado: 3-18-9-16-20-15-7-18-1-6-9-1																									

Figura 10.11 – Tabela de cifragem por substituição
Fonte: Elaborada pelo autor (2020)

Observe-se que a ordem dos símbolos de substituição (figura acima), no caso, os números de 1 a 26 podem assumir qualquer ordem desejada: quanto mais aleatória, maior a força da cifragem.

10.5.2 cifra de César

A cifra de César é uma cifra de substituição monoalfabética na qual cada letra do texto claro é substituída por outra com deslocamento predeterminado, por exemplo, considere-se um deslocamento de 3, isto significa que todas as letras “a” da mensagem deverão ser substituídas pela letra “d”, todas as letras “b” da mensagem deverão ser substituídas pela letra “e” e assim sucessivamente (figura abaixo).

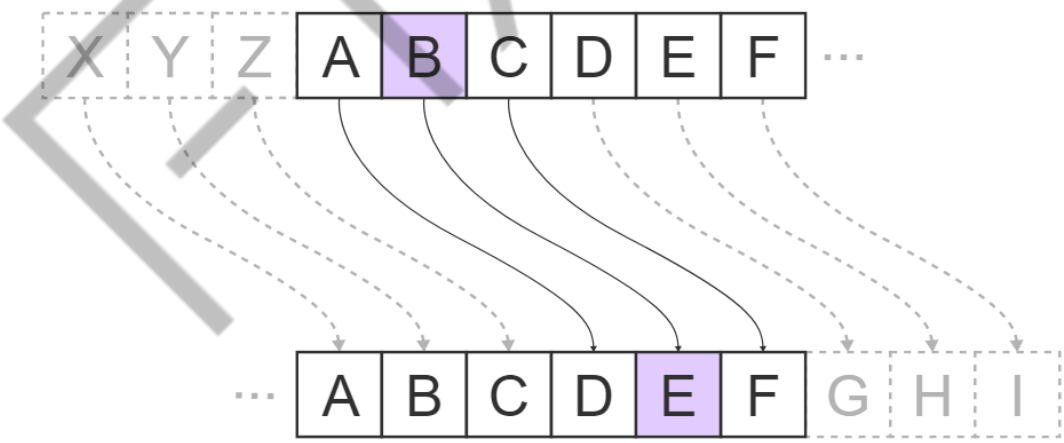

Figura 10.12 – Cifra de César
Fonte: Elaborada pelo autor (2020)

10.5.3 Cifra de Vigenère

A cifra de Vigenère é uma cifra de substituição polialfabética que tem como base a cifra de César. A principal diferença entre elas recai sobre a forma como os

caracteres são deslocados para cifragem da mensagem. Enquanto na cifra de César todos os elementos de uma mensagem têm o mesmo deslocamento, na cifra de Vigenère, o deslocamento de cada caractere é dado com base em uma senha - a chave criptográfica utilizada para codificação, e na posição do caractere em relação à mensagem. A figura abaixo ilustra a tabela polialfabética utilizada pela cifra de Vigenère.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figura 10.13 – Quadrado de Vigenère
Fonte: Elaborada pelo autor (2020)

Para codificação da mensagem “criptografia simétrica”:

- Deve-se escolher a chave criptográfica a ser utilizada para mapeamento da mensagem no quadrado de Vigenère: urano.
- Igualar o comprimento da chave ao comprimento da mensagem:

c	r	i	p	t	o	g	r	a	f	i	a	s	i	m	e	t	r	i	c	a
u	r	a	n	o	u	r	a	n	o	u	r	a	n	o	u	r	a	n	o	u

Figura 10.14 – Cifra de Vigenerè
Fonte: Elaborada pelo autor (2020)

Da correspondência estabelecida entre cada caractere da mensagem em texto claro e a chave criptográfica fornecida (tabela), obtém-se o par a ser utilizado para consulta ao quadrado de Vigenère e cifragem da mensagem.

Mensagem (linha)	c	r	i	p	t	o	g	r	a	f	i	a	s	i	m	e	t	r	i	c	a
Chave (coluna)	u	r	a	n	o	u	r	a	n	o	u	r	a	n	o	u	r	a	n	o	u
Caractere obtido no quadrado de Vigenére	w	i	i	c	h	i	x	r	n	t	c	r	s	v	a	y	k	r	v	q	u

Figura 10.15 – Cifragem da mensagem por Vigenère
Fonte: Elaborada pelo autor (2020)

Da tabela vem que a mensagem cifrada é **wiichixrntcrsvaykrvqu**.

10.5.4 cifras de transposição

Neste tipo de cifra, a mensagem é embaralhada por meio da reordenação lógica dos símbolos da mensagem em texto claro, em vez de sua substituição. A figura “Cifragem de transposição” apresenta uma tabela que mostra uma cifra de transposição de colunas, uma das mais comuns, que se baseia em uma chave que consiste de uma palavra ou frase que não contenha letras repetidas.

O objetivo da chave é numerar as colunas de modo que a coluna 1 fique posicionada abaixo da letra da chave mais próxima ao início do alfabeto e assim sucessivamente (linha 2). A mensagem em texto claro é escrita horizontalmente (em linhas) – sem espaços, enquanto o texto cifrado é lido verticalmente (em colunas), a partir da coluna cuja letra da chave seja a mais baixa.

M	I	L	K	W	A	Y
5	2	4	3	6	1	7
a	c	r	i	p	t	o
g	r	a	f	i	a	q
a	r	a	n	t	e	a
c	o	n	f	i	d	e
n	c	i	a	l	i	d
a	d	e	d	a	m	a
n	s	a	g	e	m	e
n	v	i	a	d	a	a
o	d	e	s	t	i	o
a	t	a	r	i	o	a

Figura 10.16 – Cifragem de transposição
Fonte: Elaborada pelo autor (2020)

Tome-se como exemplo, a mensagem: “a criptografia garante a confidencialidade e a autenticidade da mensagem enviada ao destinatário”, a ser cifrada com a chave MILKWAY – vide figura acima, com base na qual, obtém-se a mensagem cifrada:

Taedimmaiocrrocdsvdtifnfadgasrraani
eaieaagacnannoapitilaedtiogaedeeana

10.5.5 Base64

Base64 é um método para codificação de dados para transferência na Internet, tendo recebido esse nome em alusão ao conjunto de 64 caracteres ([A-Z],[a-z],[0-9], "/" e "+") que o compõe. O carácter "=" é utilizado como um sufixo especial, tendo a especificação original (RFC 989) definido que o símbolo "*" poderá ser utilizado para delimitar dados convertidos, mas não criptografados, dentro de um stream.

O *shell script* ilustrado pela figura abaixo demonstra de forma bastante simples a utilização da ferramenta ***base64*** do Linux, para codificação e decodificação de mensagens em base64.

```
 1 #!/bin/bash
 2 #
 3 clear
 4 echo -e "________ BASE64 _________\n"
 5 echo "Opcoes"
 6 echo "1. Codificar input em base64"
 7 echo "2. Decodificar input em base64"
 8 echo -e "3. Sair\n"
 9 read -p ">> " opt
10 #
11 case $opt in
12   1) read -p "Input a codificar: " codinp
13      echo $codinp | base64
14      read -p "Tecle algo para recomecar" i
15      ./codif64.sh ;;
16      #
17   2) read -p "Input a decodificar: " decodinp
18      echo $decodinp | base64 -d
19      read -p "Tecle algo para recomecar" i
20      ./codif64.sh ;;
21      #
22   3) echo "Script finalizado" ;;
23 esac
```

Figura 10.17 – *Shell script* para codificação/decodificação em base64
Fonte: Elaborada pelo autor (2020)

Ao ser executado, o *shell script* ilustrado na figura acima solicita ao usuário que forneça a mensagem a ser codificada (linhas 12 a 15), na linha 13 observa-se que a saída do comando *echo* é redirecionada, por meio do *pipe* (“|”), para a entrada do comando *base64* fazendo com que o *input* fornecido pelo usuário do *script* seja codificado (veja figura Codificação do *input* do usuário).

```
________ BASE64 _________

Opcoes
1. Codificar input em base64
2. Decodificar input em base64
3. Sair

>> 1
Input a codificar: codificacao64
Y29kaWZpY2FjYW82NAo=
Tecle algo para recomecar
```

Figura 10.18 – Codificação do *input* do usuário
Fonte: Elaborada pelo autor (2020)

Da figura acima é possível observar que a *string* “codificacao64” foi codificada pelo *script*, apresentando na saída o resultado **Y29kaWZpY2FjYW82NAo=**. Na linha 18 do *script* é possível observar que o comando *base64* vem acompanhado pela opção *-d*, a qual indica que a entrada do usuário foi feita em *base64* e deve ser decodificada.

Mais informações a respeito do comando *base64* poderão ser obtidas diretamente no *prompt* do Linux, digitando-se: ***man base64***. Nas linhas 15 e 20, ./codif64.sh é o nome atribuído ao *script*. Tendo, portanto, essas linhas como função, reiniciá-lo assim que o usuário digitar algo conforme solicitado após a codificação ou decodificação da mensagem.

10.6 Hands On

Normalmente, a administração de servidores não é feita interativamente, ou seja, o administrador não acessa o ativo diretamente através do console dele, mas sim, remotamente.

Uma das opções mais simples e seguras para acesso remoto a servidores Linux é via SSH, já que este se utiliza de criptografia forte para proteção da sessão. A implementação do SSH em hosts Windows pode ser feita de forma também bastante simples, por meio do PUTTY, uma aplicação *open-source* bastante leve e intuitiva.

Este *hands on* tem como objetivo demonstrar uma forma de administrar remotamente servidores Linux a partir de um *host* Windows, promovendo a autenticação no *host* remoto (servidores a serem acessados) unicamente através da chave pública do administrador, demonstrando, assim, uma aplicação prática do algoritmo de Diffie-Hellman em ambiente de produção corporativo.

Nesse novo ambiente de testes, os *hosts* Debian1 e Debian2 representam os servidores a serem remotamente administrados, sendo ainda, que as três máquinas virtuais do cenário deverão ter as respectivas interfaces de rede configuradas no VirtualBox como REDE NAT.

Vale reforçar ainda, que os endereços IP atribuídos aos *hosts* poderão variar, diferenciando-se dos indicados na figura abaixo.

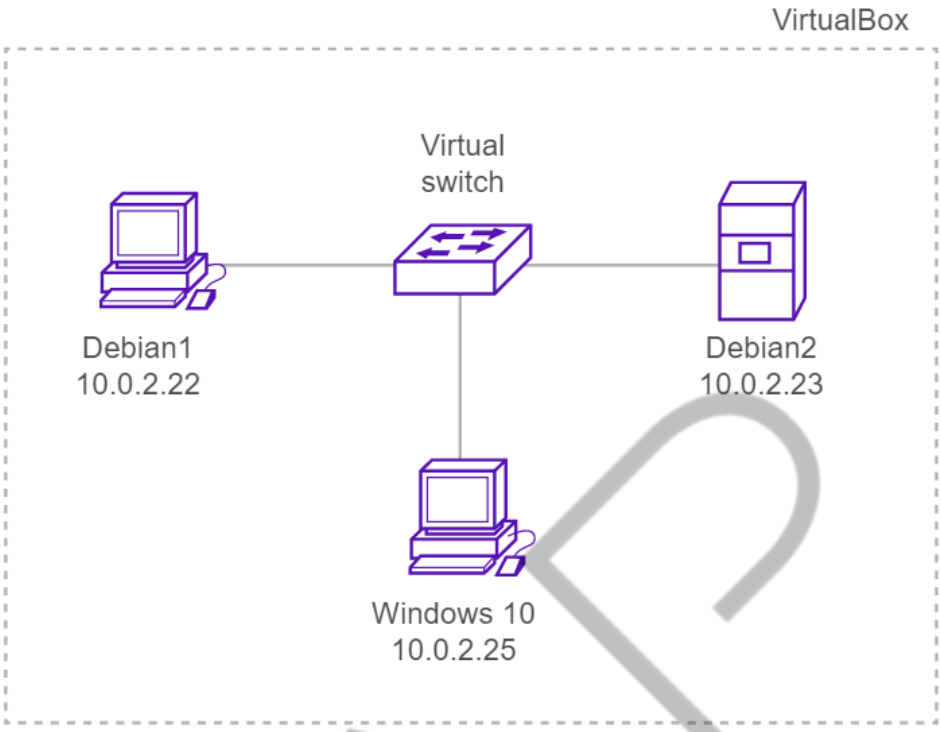

Figura 10.19 – Cenário para testes
Fonte: Elaborada pelo autor (2020)

Inicialmente, o OpenSSH-Server deverá ser instalado nos hosts Linux. Para tal, com privilégios de *root*, executa-se a linha de comando:

```
#apt-get update && apt-get install -y openssh-server
```

Agora, a partir do host Windows 10, o Putty deverá ser baixado e instalado. Para estabelecer uma conexão SSH com os *hosts* Linux, basta lançar o Putty e digitar o endereço IP do *host* Linux de destino no campo “Host Name (or IP address)”, como mostrado na figura a seguir.

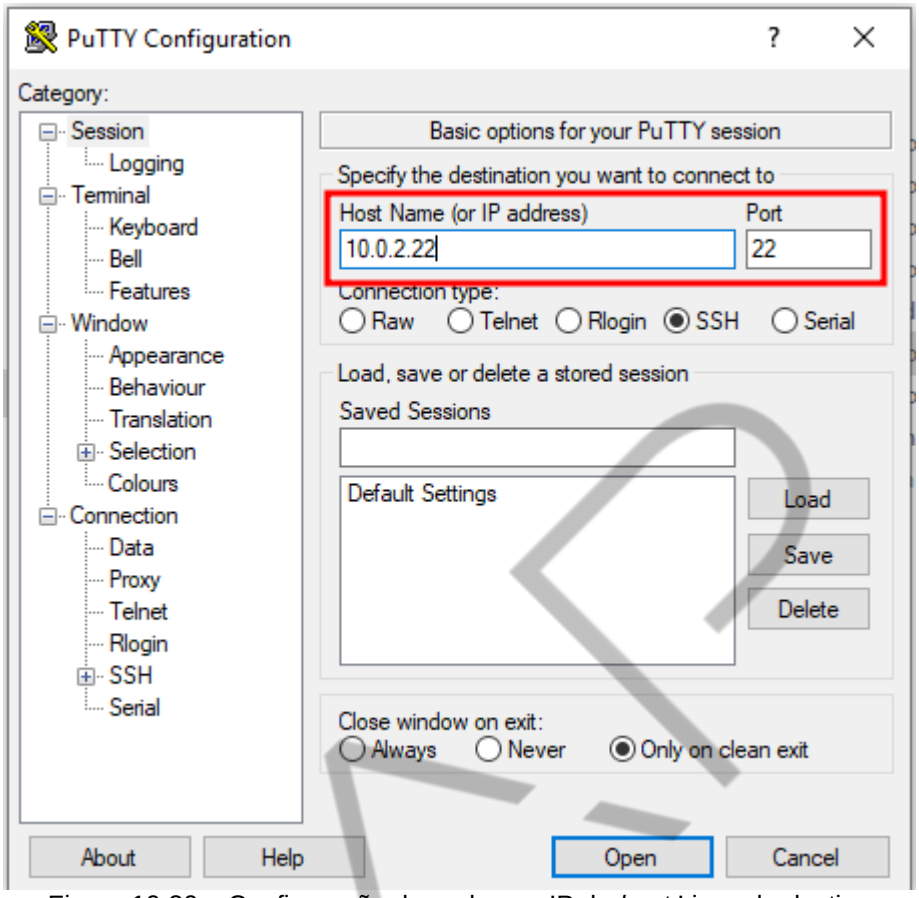

Figura 10.20 – Configuração do endereço IP do *host* Linux de destino
Fonte: Elaborada pelo autor (2020)

A clicar-se no botão Open (figura acima) um terminal texto será aberto para conexão ao *host* de destino especificado, sendo então solicitadas as credenciais a serem usadas para autenticação do usuário. Serão fornecidas as credenciais do User1 (figura abaixo), cuja conta já foi previamente configurada em ambos os *hosts* Linux (Debian01 e Debian02).

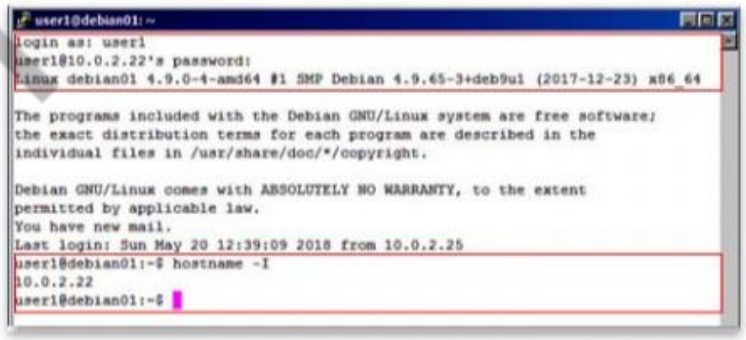

Figura 10.21 – Terminal para acesso remoto ao *host* Debian01
Fonte: Elaborada pelo autor (2020)

Por meio da figura “Terminal para acesso remoto ao *host* Debian01”, vê-se que as credenciais do usuário foram solicitadas, e que após devidamente validadas,

o acesso remoto foi permitido. O acesso ao *host* Debian2 será feito de forma diferente. Serão geradas as **chaves pública e privada** do *host* Windows sendo, então, a **chave pública** devidamente transferida e "instalada" no *host* de destino. Isto permitirá que o *host* Windows estabeleça uma conexão segura com o *host* Linux (Debian2), porém agora, de forma automatizada e sem qualquer intervenção do administrador (ou do *root*), o que pode ser muito útil, por exemplo, quando há necessidade de execução de algum *script* no *host* de destino. A geração das chaves é feita por meio do aplicativo *puttygen.exe*, que acompanha o Putty quando da instalação dele.

A figura abaixo ilustra a *interface* do *puttygen.exe*.

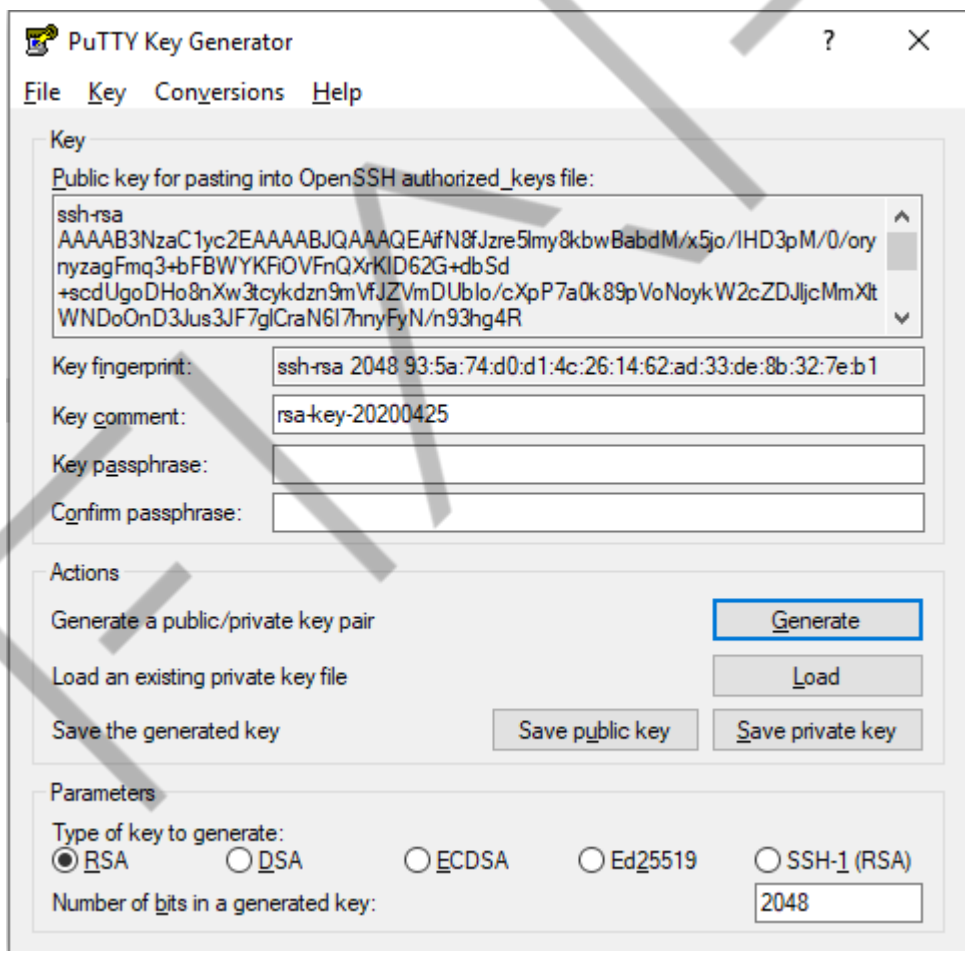

Figura 10.22 – Geração das chaves pelo *puttygen*
Fonte: Elaborada pelo autor (2020)

Para geração das chaves, basta clicar sobre o botão "Generate" (figura Geração das chaves pelo *puttygen*) e mover o *mouse* o mais rápido e desordenadamente possível. Isto é necessário para que a chave gerada tenha maior aleatoriedade e, portanto, seja mais segura.

Ainda por intermédio dessa figura, é possível verificar que as chaves foram geradas com base no algoritmo RSA, com tamanho de 2048 *bits*, comprimento padrão para esse algoritmo. Para salvar a chave pública, deve-se clicar com o botão direito sobre o conteúdo da caixa na parte superior da figura “Terminal para acesso remoto ao *host* Debian01”, selecionar a opção Select All, finalmente, clicar sobre a opção Copy. Esse conteúdo deverá ser colado em um arquivo texto (puro), nomeado e armazenado em local adequado. Sugere-se nomear o arquivo como **id_rsa.pub**.

Não utilizar o botão **Save public key** disponível no painel do Putty para salvar a **chave pública**, pois ela será incompatível com o formato utilizado pelo Linux.

Salva a chave pública, a chave privada deverá ser gravada por meio do botão Save private key, no mesmo diretório onde a chave pública foi gravada, sugerindo-se que ela seja nomeada apenas como **id_rsa** (o próprio Putty colocará a extensão .ppk).

A partir do terminal do *host* Windows a chave pública deverá ser copiada para o *host* Debian02 (figura abaixo), por intermédio do aplicativo *pscp* que também acompanha o Putty.

Figura 10.23 – Transferência da chave pública com o aplicativo *pscp*
Fonte: Elaborada pelo autor (2020)

Da figura “Geração das chaves pelo *puttygen”* é possível ver, por meio do aplicativo *pscp* o arquivo id_rsa.pub foi copiado para o diretório de trabalho do usuário user1 no *host* Debian2. No diretório de trabalho do *user1* no *host* Debian02, a chave pública do *host* Windows deverá ser “instalada” de acordo com a linha:

```
user1@debian02:~$ cat id_rsa.pub >> .ssh/authorized_keys
```

Essa linha fará a inserção da chave pública no arquivo *.ssh/authorized_keys*, criando o arquivo, caso ele não exista. Cabe ressaltar que o uso dos dois redirecionadores (>>) indicados na referida linha de comando, serve para evitar que o conteúdo do arquivo venha a ser sobrescrito, caso já exista.

O arquivo de configuração do servidor SSH *(/etc/ssh/sshd_config*) deve ser ajustado agora, de forma a não mais aceitar a autenticação dos clientes via senha, mas sim, via chave pública. A figura abaixo exibe os ajustes a serem feitos.

Configuração original	Configuração ajustada
#PasswordAuthentication yes	PasswordAuthentication no
PubkeyAuthentication yes	Manter se já estiver assim, ajustar em caso contrário

Figura 10.24 – Ajustes na configuração do servidor SSH
Fonte: Elaborada pelo autor (2020)

Após os devidos ajustes, o serviço deverá ser reinicializado:

```
#systemctl restart sshd.service
```

Por fim, faltam os ajustes finais do Putty a fim de habilitar o acesso ao *host* remoto somente com as chaves pública e privada. Inicialmente será criada uma sessão para acesso ao *host* remoto (figura abaixo).

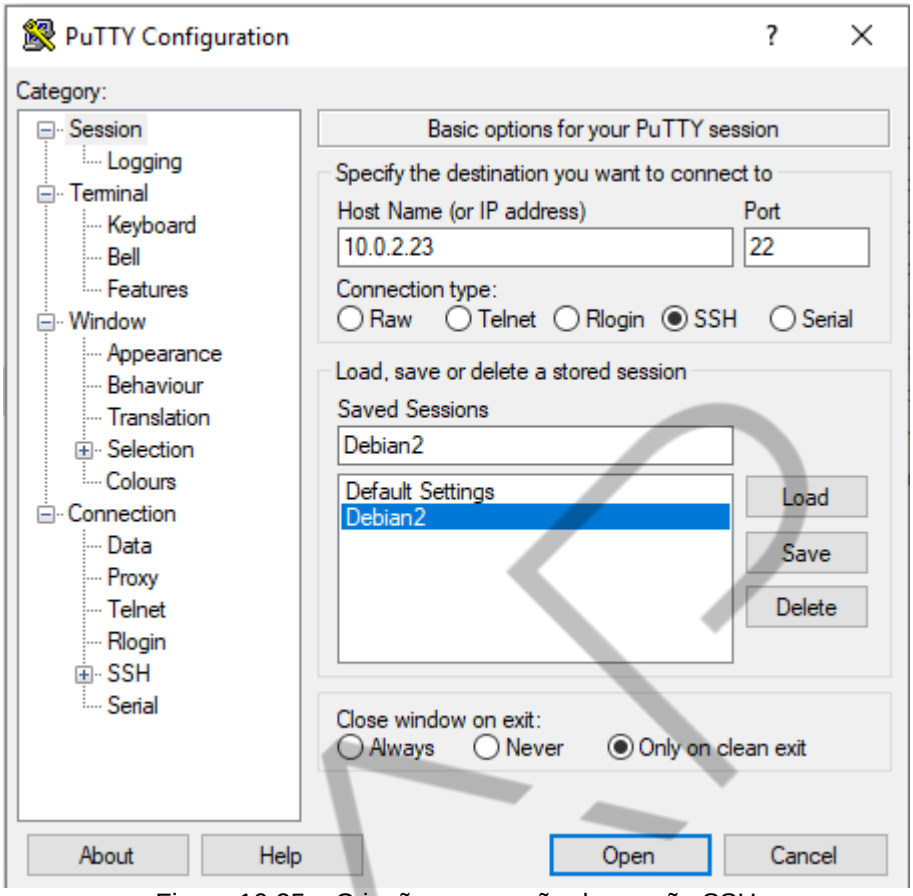

Figura 10.25 – Criação e gravação da sessão SSH
Fonte: Elaborada pelo autor (2020)

Na página principal do Putty (figura Criação e gravação da sessão SSH), o campo Host Name (ou IP address) deverá ser preenchido com o endereço IP do *host* Debian02: 10.0.2.23. A seguir, o campo Saved Sessions deverá ser preenchido com o nome do *host*: Debian02, salvando-se então a sessão por meio do botão Save.

A partir do painel esquerdo, clicar em Connection -> Data, preenchendo o campo Auto-login username com user1 (figura abaixo).

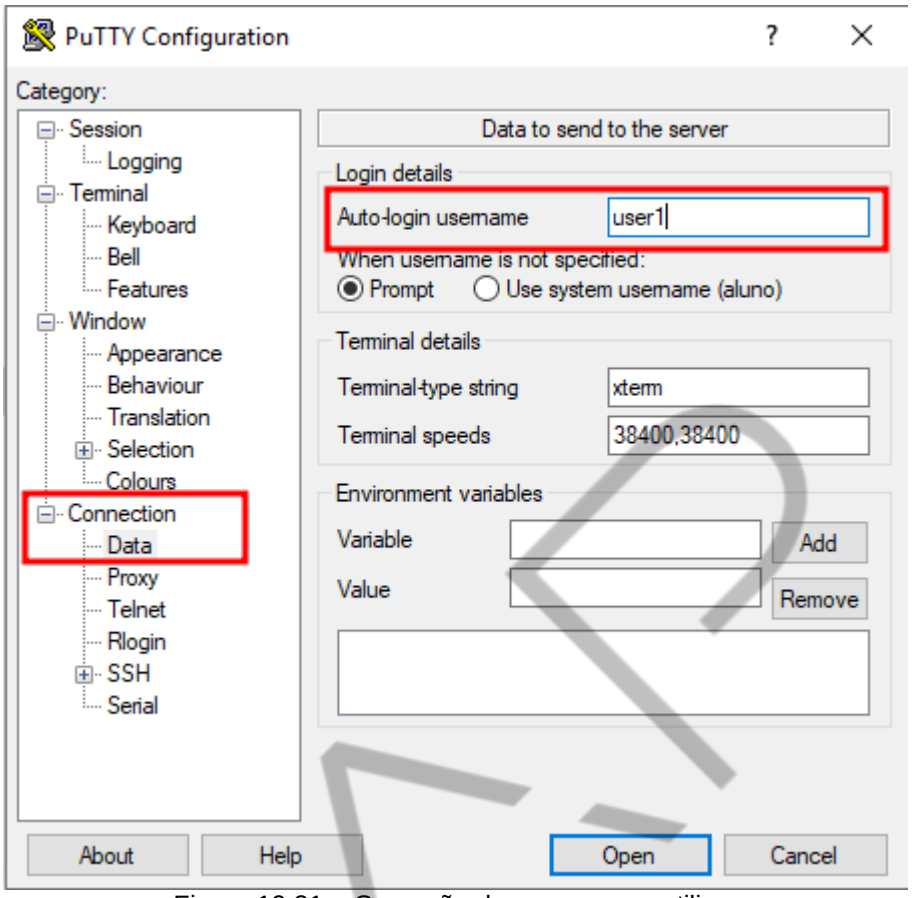

Figura 10.21 – Gravação do *username* a utilizar
Fonte: Elaborada pelo autor (2020)

Finalmente, deve ser configurada ainda a autenticação por chave, agora, a chave privada. Para tal, ainda a partir do painel esquerdo do Putty, deve-se acessar SSH->Auth, e preencher o campo Private key file or authentication (figura Gravação do username a utilizar) fornecendo o arquivo da chave privada.

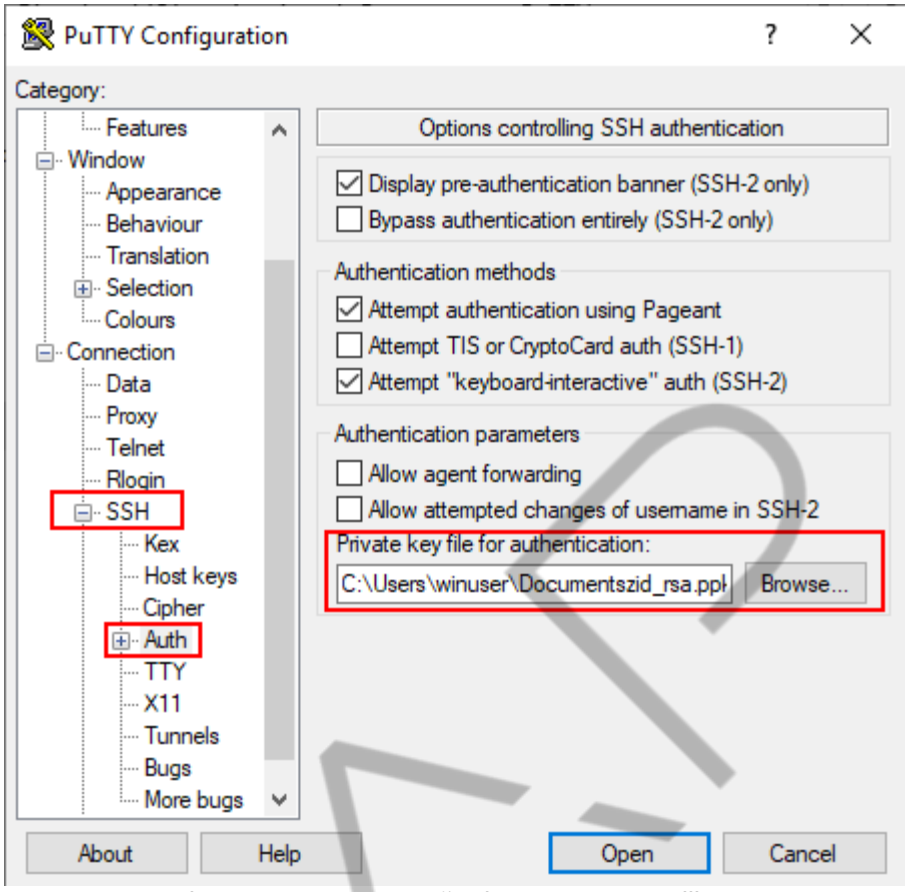

Figura 10.21 – Gravação do username a utilizar
Fonte: Elaborada pelo autor (2020)

Para conexão ao *host* remoto, basta agora clicar no botão Open (figura Gravação do username a utilizar).

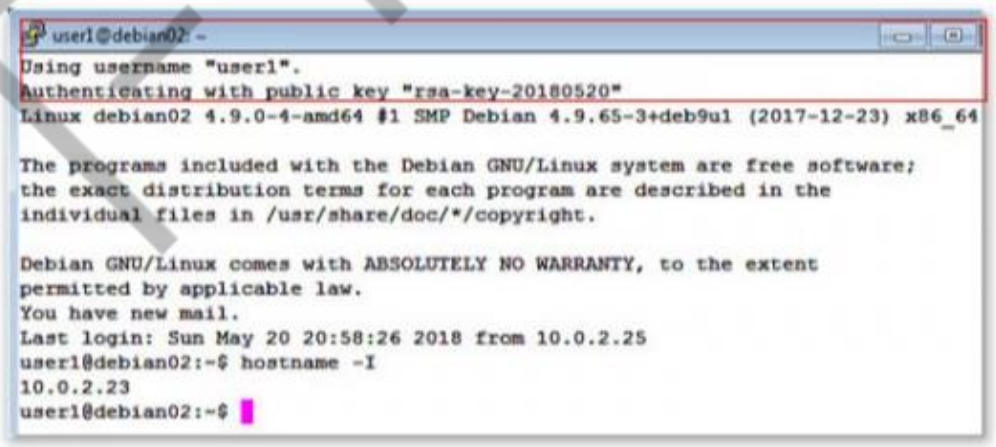

Figura 10.23 – Gravação do username a utilizar
Fonte: Elaborada pelo autor (2020)

Na figura “Gravação do username a utilizar”, é possível observar a mensagem de autenticação com base na chave pública, e a confirmação do acesso ao *host* remoto, Debian02. Vale ressaltar ser esse recurso bastante útil quando, por

exemplo, da necessidade de execução de tarefas em um *host* remoto de forma automatizada e sem intervenção do administrador ou do *root*. Entretanto, não se recomenda seu uso indiscriminado.

REFERÊNCIAS

CABRAL, C.; CAPRINO, W. Trilhas em segurança da informação. Caminhos e ideias para proteção de dados. Rio de Janeiro: Brasport, 2016.

FIPS. **Secure Hash Standard (SHS).** [s.d.]. Disponível em <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>. Acesso em: 25 abr. 2020.

GTA UFRJ. **SSL & TLS.** Disponível em: <https://www.gta.ufrj.br/grad/11_1/tls/>. Acesso em: 25 abr. 2020.

IUS MENTIS. **Message digest / cryptographic hash functions.** [s.d.]. Disponível em: <http://www.iusmentis.com/technology/hashfunctions/>. Acesso em: 25 abr. 2020.

MORAES, A.F. **Segurança em Redes**. Fundamentos. São Paulo: Érica. 2010.

OFICIOELETRONICO. **Cartilha de Certificação Digital.** [s.d.]. Disponível em: <https://www.oficioeletronico.com.br/Downloads/CartilhaCertificacaoDigital.pdf>. Acesso em: 25 abr. 2020.

QUALISIGN. **O que significa Não-Repúdio.** [s.d.]. Disponível em: <https://www.documentoeletronico.com.br/o-que-significa-nao-repudio.asp>. Acesso em: 25 abr. 2020.

TENEMBAUM, A. S. **Redes de Computadores**. e-book. 5. ed. São Paulo: Pearson, 2016.

WALKER, M. **CEH Certified Ethical Hacker**. Exam Guide. USA: McGraw Hill, 2012.