sshenc no Windows

Tutorial simples para proteger chaves SSH com TPM 2.0 e Windows Hello

Guia prático

7 de outubro de 2026

Sumário

O que é o sshenc **2**

Antes de começar **3**
Verificar o TPM 3
Verificar o Windows Hello 3
Verificar o OpenSSH 3

1. Baixar o sshenc **4**
Atenção ao SmartScreen 4
Verificação opcional do SHA-256 4

2. Instalar **4**
O que `sshenc install` configura no Windows 5

3. Criar sua primeira chave protegida pelo TPM **5**
Por que usar um label? 5

4. Conferir a chave criada **6**

5. Copiar a chave pública no Windows **6**

6. Autorizar a chave em um servidor Debian/Linux **6**

7. Testar o acesso ao servidor **6**
Usar `ssh` normalmente 7

8. Usar com GitHub **7**

9. Usar com Git **7**

10. Commits "Verified" no GitHub **8**

11. Vários computadores **8**

12. Git Bash **9**

13. VS Code, Cursor e Windows Terminal **9**

14. WSL **9**

15. Ver o que o sshenc tem configurado **10**
Listar chaves 10
Inspecionar uma chave 10
Exibir a chave pública 10
Ver o arquivo de configuração do OpenSSH 10
Ver qual SSH o Git está usando 10

16. Diagnóstico quando algo não funciona **10**
Testar o SSH com log detalhado 10
Rodar o agente com log de debug 10
Reaplicar a configuração 11
Confirmar que a chave existe 11

17. Apagar uma chave **11**

18. Desinstalar **11**

19. Minha configuração recomendada **11**

20. O que não fazer **12**

Resumo rápido **12**

Fontes consultadas **12**

O que é o sshenc

O **sshenc** é um gerenciador de chaves SSH protegidas por hardware. No Windows, ele cria a chave privada dentro do **TPM 2.0** do computador e a disponibiliza ao OpenSSH por meio de um agente.

A ideia principal é esta:

```
Windows / Git / SSH
        |
        v
   sshenc-agent
        |
        v
 Windows Hello
        |
        v
     TPM 2.0
        |
        v
 chave privada não exportável
```

A chave privada não fica salva como um arquivo comum que possa ser copiado de `~/.ssh`. O Windows mantém a chave dentro do TPM. No disco ficam apenas metadados e cópias da **chave pública**.

Neste tutorial vamos usar a configuração que considero mais segura para uso pessoal: **exigir Windows Hello a cada uso da chave**.

> **Importante:** uma chave criada no TPM fica vinculada àquele computador. Ela não

pode ser exportada, clonada ou restaurada em outro PC. Em vários computadores, crie uma chave diferente em cada um e cadastre todas as chaves públicas nos seus servidores e serviços.

Antes de começar

Você precisa de:

- Windows com TPM 2.0 ativo;
- Windows Hello configurado, de preferência PIN + biometria quando disponível;
- Windows OpenSSH Client;
- acesso administrativo apenas se precisar instalar o OpenSSH Client.

Verificar o TPM

Abra o **PowerShell** e execute:

```
Get-Tpm | Select-Object TpmPresent,TpmReady,TpmEnabled,TpmActivated
```

O ideal é aparecer algo semelhante a:

```
TpmPresent   : True
TpmReady     : True
TpmEnabled   : True
TpmActivated : True
```

Se `TpmPresent` ou `TpmReady` estiverem como `False`, resolva isso antes de criar as chaves.

Você também pode abrir:

```
tpm.msc
```

para consultar o estado do TPM pela interface do Windows.

Verificar o Windows Hello

No Windows, vá em:

```
Configurações
  -> Contas
  -> Opções de entrada
```

Configure pelo menos um **PIN do Windows Hello**. Impressão digital e reconhecimento facial também podem ser usados se o computador oferecer suporte.

Verificar o OpenSSH

No PowerShell:

```
ssh -V
```

Se aparecer a versão do OpenSSH, está pronto.

Se o comando não existir, abra um PowerShell como Administrador e verifique o recurso:

```
Get-WindowsCapability -Online |
    Where-Object Name -like 'OpenSSH.Client*'
```

Se estiver como `NotPresent`, instale:

```
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
```

Depois feche e abra o terminal novamente.

1. Baixar o sshenc

A forma mais simples no Windows Intel/AMD 64-bit é usar o instalador MSI oficial da página de releases:

GitHub - sshenc - Latest Release

Baixe o arquivo:

```
sshenc-x86_64-pc-windows-msvc.msi
```

Na data deste guia, a release publicada mais recente é a **v0.6.101**.

Atenção ao SmartScreen

Os binários de Windows publicados pelo projeto são compilados pelo GitHub Actions, mas atualmente **não possuem assinatura Authenticode**. Por isso o Windows Defender SmartScreen pode mostrar um aviso.

Isso não significa automaticamente que o arquivo seja malicioso, mas não ignore o aviso sem conferir a origem do arquivo.

Baixe apenas da página oficial:

```
https://github.com/godaddy/sshenc/releases/latest
```

Verificação opcional do SHA-256

Para a release **v0.6.101**, o MSI x86_64 publicado pelo projeto possui o seguinte SHA-256:

```
5342426bbbbbf868800d522b47f4a390662f05860f67ee5bb14e27e038e65b30
```

No PowerShell, dentro da pasta onde o MSI foi baixado:

```
Get-FileHash .\sshenc-x86_64-pc-windows-msvc.msi -Algorithm SHA256
```

Compare o valor apresentado.

> Se você estiver usando uma versão diferente da v0.6.101, não compare com o hash acima. Consulte o artefato da release correspondente.

2. Instalar

Execute o MSI normalmente.

O instalador faz duas coisas importantes automaticamente:

1. adiciona o `sshenc` ao PATH do Windows;
2. executa `sshenc install` para configurar o agente SSH.

Depois da instalação, **feche e abra novamente o PowerShell ou Windows Terminal**.

Teste:

```
sshenc --version
```

E depois:

```
sshenc list
```

Se ambos responderem normalmente, a instalação básica está pronta.

O que `sshenc install` configura no Windows

Segundo a documentação do projeto, o comando:

```
sshenc install
```

faz automaticamente o seguinte:

- configura o `IdentityAgent` do OpenSSH para usar o named pipe do Windows;
- define `GIT_SSH_COMMAND` para o OpenSSH nativo do Windows;
- inicia o `sshenc-agent` em segundo plano;
- detecta instalações WSL e tenta configurá-las.

O named pipe usado pelo agente é:

```
\\.\pipe\openssh-ssh-agent
```

Você normalmente não precisa configurar isso manualmente.

3. Criar sua primeira chave protegida pelo TPM

Vou usar o nome `servidores` como exemplo.

No PowerShell:

```
sshenc keygen --label servidores --require-user-presence
```

O parâmetro mais importante é:

```
--require-user-presence
```

Ele faz o sshenc configurar a chave para exigir confirmação do usuário. No Windows, isso significa **Windows Hello** - PIN, impressão digital ou reconhecimento facial, de acordo com o que estiver configurado no PC.

Assim, quando algum programa tentar usar essa chave para assinar uma autenticação SSH, o Windows deverá pedir sua autorização.

Por que usar um label?

O `label` é apenas um nome para identificar a chave.

Exemplos bons:

```
servidores
portalidea
github-pessoal
github-trabalho
vps
```

Evite espaços e nomes confusos.

4. Conferir a chave criada

Liste as chaves:

```
sshenc list
```

Para ver detalhes da chave `servidores`:

```
sshenc inspect servidores
```

Para mostrar a chave pública:

```
sshenc export-pub servidores
```

A chave **pública** pode ser copiada, enviada e cadastrada em servidores sem problema. A chave privada continua protegida pelo TPM.

5. Copiar a chave pública no Windows

No PowerShell, use:

```
sshenc export-pub servidores | Set-Clipboard
```

Pronto. A chave pública estará na área de transferência.

Isso substitui o `pbcopy` mostrado em alguns exemplos do README, que é um comando de macOS.

6. Autorizar a chave em um servidor Debian/Linux

Entre no servidor usando o método que você já usa hoje.

No servidor:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
nano ~/.ssh/authorized_keys
```

Cole a chave pública copiada do Windows em uma nova linha, salve e saia.

Depois:

```
chmod 600 ~/.ssh/authorized_keys
```

Não remova sua chave SSH antiga ainda. Primeiro teste a nova em outra janela de terminal.

7. Testar o acesso ao servidor

No Windows, para obrigar o sshenc a usar a chave chamada `servidores`:

```
sshenc ssh --label servidores usuario@servidor
```

Exemplo:

```
sshenc ssh --label servidores root@192.168.50.120
```

Se tudo estiver correto:

1. o SSH tentará autenticar;
2. o Windows Hello será exibido;
3. você autoriza com PIN, biometria ou reconhecimento facial;
4. o TPM realiza a operação criptográfica;
5. a conexão SSH é aberta.

A chave privada não é entregue ao `ssh.exe`.

Usar `ssh` normalmente

Depois que o agente estiver configurado, o comando tradicional também pode funcionar normalmente:

```
ssh usuario@servidor
```

Se você tiver várias chaves, o wrapper com `--label` é mais previsível:

```
sshenc ssh --label servidores usuario@servidor
```

8. Usar com GitHub

Você pode criar uma chave separada para o GitHub:

```
sshenc keygen --label github --require-user-presence
```

Copie a pública:

```
sshenc export-pub github | Set-Clipboard
```

No GitHub, abra:

```
Settings
  -> SSH and GPG keys
  -> New SSH key
```

Escolha **Authentication Key**, dê um nome ao computador e cole a chave.

Exemplo de nome:

```
POE - Windows TPM
```

Depois teste:

```
sshenc ssh --label github -T git@github.com
```

Na primeira conexão, confirme a fingerprint do GitHub se o OpenSSH perguntar.

9. Usar com Git

O projeto inclui o comando `gitenc`.

Para clonar usando uma identidade específica:

```
gitenc --label github clone git@github.com:USUARIO/REPOSITORIO.git
```

Dentro de um repositório existente, você pode fixar a identidade:

```
cd C:\caminho\do\repositorio
gitenc --config github
```

Depois disso, você usa Git normalmente:

```
git pull
git push
git commit -m "minha alteração"
```

O `gitenc --config github` configura o repositório para usar aquela identidade e também prepara assinatura de commits via SSH.

10. Commits "Verified" no GitHub

Se quiser assinar commits com a mesma chave TPM:

1. copie novamente a chave pública:

```
sshenc export-pub github | Set-Clipboard
```

2. no GitHub, abra `Settings -> SSH and GPG keys`;
3. clique em `New SSH key`;
4. escolha **Signing Key**;
5. cole a mesma chave pública;
6. no repositório, execute:

```
gitenc --config github
```

A partir daí, commits criados nesse repositório podem ser assinados pela chave protegida pelo TPM.

Teste:

```
git commit --allow-empty -m "teste de assinatura TPM"
git log --show-signature -1
```

Depois de enviar o commit, o GitHub deve conseguir mostrá-lo como **Verified**, desde que a Signing Key esteja cadastrada corretamente.

11. Vários computadores

Este ponto é fundamental.

Não tente copiar a chave do sshenc para outro computador. A não exportabilidade é justamente uma das proteções do sistema.

Faça assim:

```
PC principal       -> chave servidores-pc
Notebook           -> chave servidores-notebook
PC secundário      -> chave servidores-secundario
```

Em cada máquina:

```
sshenc keygen --label servidores-pc --require-user-presence
```

Use um nome adequado àquela máquina.

Depois adicione as três **chaves públicas** ao `authorized_keys` dos servidores.

O servidor pode aceitar várias chaves públicas para a mesma conta sem problema.

Se uma máquina for perdida, formatada ou aposentada, basta remover a chave pública correspondente do servidor.

12. Git Bash

O Git for Windows normalmente traz seu próprio `ssh.exe` baseado em MINGW. Esse SSH não entende o named pipe utilizado pelo agente do sshenc.

Por isso o `sshenc install` define automaticamente:

```
GIT_SSH_COMMAND=C:\Windows\System32\OpenSSH\ssh.exe
```

Assim, os comandos Git passam a usar o OpenSSH nativo do Windows.

Para conferir no PowerShell:

```
$env:GIT_SSH_COMMAND
```

Se necessário, feche e abra novamente o terminal após a instalação.

13. VS Code, Cursor e Windows Terminal

A documentação do projeto indica suporte aos terminais do VS Code e ao Windows Terminal. Como o sshenc integra com o OpenSSH do Windows, os programas que usam esse SSH herdam a configuração do agente.

No Cursor ou VS Code, teste pelo terminal integrado:

```
sshenc list
```

Depois:

```
sshenc ssh --label servidores usuario@servidor
```

Para Git dentro do editor, a configuração feita pelo `gitenc` no repositório também continua valendo.

14. WSL

O sshenc possui suporte específico para WSL.

Ao executar:

```
sshenc install
```

no Windows, ele tenta detectar as distribuições WSL e criar uma ponte entre o socket Unix do Linux e o named pipe do Windows.

A documentação informa que essa integração usa `socat` e `npiperelay`.

Depois de configurado, dentro do WSL você pode usar:

```
ssh usuario@servidor
git push
```

```
scp arquivo usuario@servidor:
sftp usuario@servidor
```

Se você instalar uma nova distribuição WSL depois, execute novamente no Windows:

```
sshenc install
```

O comando foi projetado para ser idempotente.

15. Ver o que o sshenc tem configurado

Listar chaves

```
sshenc list
```

Inspecionar uma chave

```
sshenc inspect servidores
```

Exibir a chave pública

```
sshenc export-pub servidores
```

Ver o arquivo de configuração do OpenSSH

```
Get-Content $HOME\.ssh\config
```

Ver qual SSH o Git está usando

```
$env:GIT_SSH_COMMAND
```

O esperado após a instalação é apontar para o OpenSSH do Windows.

16. Diagnóstico quando algo não funciona

Testar o SSH com log detalhado

```
ssh -vvv usuario@servidor
```

Ou forçando uma chave do sshenc:

```
sshenc ssh --label servidores -vvv usuario@servidor
```

O `-vvv` mostra quais chaves estão sendo oferecidas e em qual etapa a autenticação falhou.

Rodar o agente com log de debug

O projeto oferece:

```
sshenc agent --debug
```

Use isso para investigar problemas do agente.

Reaplicar a configuração

Se o agente ou a configuração do SSH parecerem incorretos:

```
sshenc install
```

O comando pode ser executado novamente.

Confirmar que a chave existe

```
sshenc list
sshenc inspect servidores
```

17. Apagar uma chave

Para remover uma chave:

```
sshenc delete servidores
```

Para também apagar os arquivos públicos associados:

```
sshenc delete servidores --delete-pub
```

> **Cuidado:** a chave privada é não exportável. Se você apagar a chave do TPM, não existe backup para restaurá-la. Antes de excluir, confirme que outra chave já funciona nos servidores e serviços necessários.

18. Desinstalar

Primeiro, você pode remover a configuração criada pelo sshenc:

```
sshenc uninstall
```

Depois remova o programa em:

```
Configurações -> Aplicativos -> Aplicativos instalados
```

O instalador MSI também é projetado para executar a limpeza durante a desinstalação.

19. Minha configuração recomendada

Para uma máquina Windows usada para administração de servidores, eu faria assim:

```
sshenc keygen --label servidores --require-user-presence
sshenc keygen --label github --require-user-presence
```

Depois:

```
sshenc list
```

E usaria:

```
sshenc ssh --label servidores root@meu-servidor
```

Para GitHub:

```
gitenc --config github
```

Essa separação tem algumas vantagens:

- uma chave para infraestrutura;
- uma chave para Git/GitHub;
- Windows Hello em cada assinatura;
- nenhuma chave privada SSH comum para essas identidades no disco;
- revogação separada caso uma identidade deixe de ser necessária.

20. O que não fazer

Evite estes erros:

- não apague suas chaves SSH antigas antes de testar o sshenc;
- não tente fazer backup da chave privada do TPM - ela foi criada para ser não exportável;
- não use o mesmo nome de chave sem saber a qual computador ele se refere;
- não desative `--require-user-presence` em chaves administrativas sem ter um motivo claro;
- não ignore um alerta do SmartScreen sem confirmar que o MSI veio da release oficial;
- não espere que PuTTY/Pageant usem o agente do sshenc; a documentação recomenda Windows OpenSSH.

Resumo rápido

Instalar pelo MSI oficial, abrir um novo PowerShell e criar a chave:

```
sshenc keygen --label servidores --require-user-presence
```

Copiar a pública:

```
sshenc export-pub servidores | Set-Clipboard
```

Adicionar ao `~/.ssh/authorized_keys` do servidor e testar:

```
sshenc ssh --label servidores usuario@servidor
```

Para GitHub:

```
sshenc keygen --label github --require-user-presence
sshenc export-pub github | Set-Clipboard
```

Cadastrar no GitHub e configurar o repositório:

```
gitenc --config github
```

Fontes consultadas

Este guia foi produzido a partir da documentação e dos arquivos atuais do projeto em 7 de outubro de 2026.

- Repositório oficial - godaddy/sshenc
- README oficial
- Política e modelo de segurança
- Release mais recente

Release verificada durante a elaboração: v0.6.101, publicada em 4 de junho de 2026.