Segurança de dados

Aula 07 – Criptografía - parte 2

Gustavo Bianchi Maia

gustavo.maia@faculdadeimpacta.com.br

Sumário

- Resumo Aula Criptografia - pt 1
- Níveis para aplicação da criptografia
  - Funções básicas
  - Always Encrypted
  - TDE (Transparent Data Encryption)

Criptografia

A criptografia tem quatro objetivos principais:

- **Confidencialidade da mensagem**: só o destinatário autorizado deve ser capaz de extrair o conteúdo da mensagem da sua forma cifrada. Além disso, a obtenção de informação sobre o conteúdo da mensagem (como uma distribuição estatística de certos caracteres) não deve ser possível, uma vez que, se o for, torna mais fácil a análise criptográfica.

- **Integridade da mensagem**: o destinatário deverá ser capaz de determinar se a mensagem foi alterada durante a transmissão.

- **Autenticação do remetente**: o destinatário deverá ser capaz de identificar o remetente e verificar que foi mesmo ele quem enviou a mensagem.

- **não-repúdio ou irretratabilidade do emissor**: não deverá ser possível ao emissor negar a autoria da mensagem.

Nem todas as técnicas garantem todos os objetivos.

Criptografia - Tipos básicos

2-WAY

Simétricos

Os algoritmos de chave simétrica (ou chave única / secreta) são uma classe de algoritmos para a criptografia, que usam chaves criptográficas relacionadas para as operações de cifragem ou decifragem (ou cifra/decifra, ou cifração/decifração).

Assimétricos (ou de chave pública/privada)

A criptografia de chave pública ou criptografia assimétrica é um método de criptografia que utiliza um par de chaves: uma chave pública e uma chave privada. A chave pública é distribuída livremente para todos os correspondentes via e-mail ou outras formas, enquanto a chave privada deve ser conhecida apenas pelo seu dono.

1-WAY

Hash

Um hash (ou escrutínio) é uma sequência de bits geradas por um algoritmo de dispersão, em geral representada em base hexadecimal, que permite a visualização em letras e números (0 a 9 e A a F), representando um nibble cada (4 bits). O conceito teórico diz que "hash é a transformação de uma grande quantidade de informações em uma pequena quantidade de informações".

Criptografia - Tipos básicos

Determinísticos

A criptografia determinística sempre gera o mesmo valor criptografado para qualquer valor de texto sem formatação. Usar a criptografia determinística proporciona pesquisas de ponto, junções de igualdade, agrupamento e indexação em colunas criptografadas. No entanto, ela também pode permitir que usuários não autorizados estimem informações sobre os valores criptografados examinando padrões na coluna criptografada, especialmente se há um conjunto pequeno de valores criptografados possíveis, como Verdadeiro/Falso ou região Norte/Sul/Leste/Oeste. A criptografia determinística deve usar uma ordenação de colunas com uma ordem de classificação binary2 para as colunas de caracteres.

Aleatórios

A criptografia aleatória usa um método que criptografa os dados de uma maneira menos previsível. A criptografia aleatória é mais segura, mas impede o uso de pesquisas, agrupamento, indexação e junção em colunas criptografadas.

fonte: Always Encrypted - SQL Server | Microsoft Docs

Criptografia - Tipos básicos

2-WAY

Simétricos

- Máquina Enigma (Máquina alemã de rotores utilizada na 2a Guerra Mundial)
- DES - Data Encryption Standard (FIPS 46-3, 1976)
- RC4 (um dos algoritmos criados pelo Prof. Ron Rivest)
- RC5 (também por Prof. Ron Rivest)
- Blowfish (por Bruce Schneier)
- IDEA - International Data Encryption Algorithm (J Massey e X Lai)
- AES (também conhecido como **RIJNDAEL**) - Advanced Encryption Standard (FIPS 197, 2001)
- RC6 (Ron Rivest)

Assimétricos (ou de chave pública/privada)

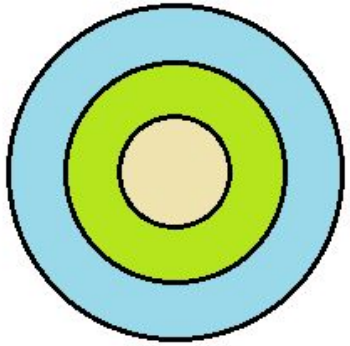
- Curvas elípticas
- Diffie-Hellman
- DSA de curvas elípticas
- El Gamal
- RSA

1-WAY

Hash

- MD5
- SHA-256
- SHA-1
- RIPEMD-160
- Tiger

Criptografia - Técnicas por nível



Criptografia de dados em repouso:

Conjunto de técnicas que visam salvaguardar o dado da forma como ele é armazenado e/ou recuperado.

Mais detalhado

- **Dado**
  - Funções internas:
    - Hashbytes
    - EncryptByKey / DecryptByKey
    - EncryptByAsymKey / DecryptByAsymKey
- **Coluna**
  - AlwaysEncrypted(Always Encrypted - SQL Server | Microsoft Docs)
- **Banco de dados**
  - Transparent Data Encryption (TDE (Transparent Data Encryption) - SQL Server | Microsoft Docs)
- **Sistema Operacional**
  - Bitlocker (BitLocker Como implantar no Windows Server 2012 e posterior - Microsoft 365 Security | Microsoft Docs)
- **Disco**
  - Full Disk Encryption (How to Enable Full-Disk Encryption on Windows 10 (howtogeek.com)
  - EBS Encrypted Volumes (Amazon EBS encryption - Amazon Elastic Compute Cloud)

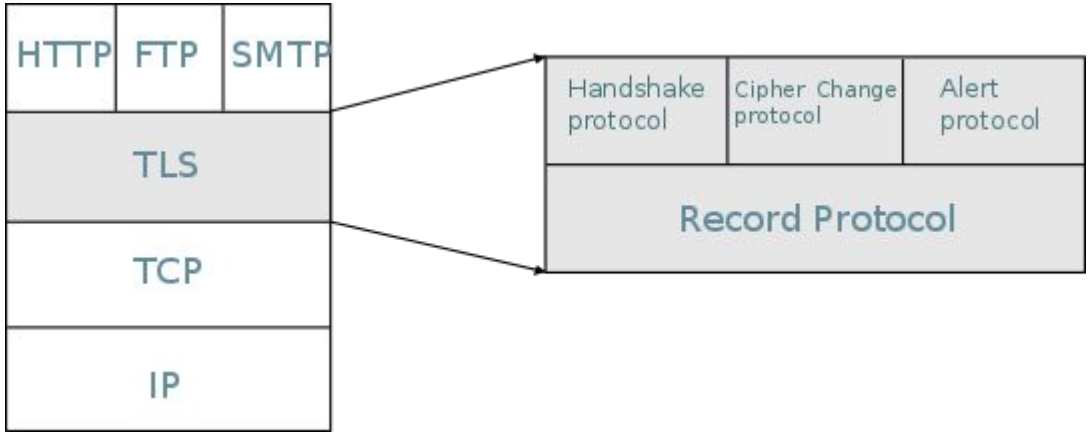
Menos detalhado

Criptografia - Técnicas por nível

Criptografia de dados em transito:

Conjunto de técnicas que visam salvaguardar o dado enquanto ele é utilizado em uma transferência ou comunicação.

SSL / TLS (Transport Layer Security – Wikipédia, a enciclopédia livre (wikipedia.org))



TLS no MSSQL (Habilitar conexões criptografadas - SQL Server | Microsoft Docs)

Criptografia - Funções internas

```
CREATE DATABASE cripto
go
USE cripto
go
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
--=X=-- HASH
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
DECLARE @Salt VARCHAR(50) = 'FIT'
DECLARE @pass VARCHAR(255) = 'teste'
DECLARE @value VARCHAR(255) = @pass + @salt
SELECT Hashbytes('md5', @value)
-- 0xF49D7CA29D54B25A20F8EE4D695F7748
SELECT Hashbytes('sha1', @value)
--0x7ECA82E54BFF01456689D9995DBD1241090FF458
```

- Sempre utilizam um algoritmo determinístico
- O tamanho do resultado pode ajudar a determinar a força do algoritmo
- Salt é um conjunto de chars adicional, adicionado à senha, para melhorar sua segurança
- Os dados criptografados salvos em colunas do tipo VARBINARY.

Criptografia - Funções internas

```
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
--=X=-- CHAVE SIMÉTRICA
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
CREATE symmetric KEY chavesimetrica01 WITH algorithm = aes_256 encryption BY
password = N'!@@QW#E#$R%dreud76'
OPEN symmetric KEY chavesimetrica01 decryption BY password =
N'!@@QW#E#$R%dreud76';

DECLARE @result VARBINARY(max)
DECLARE @key UNIQUEIDENTIFIER = (SELECT Key_guid('ChaveSimetrica01'))

SELECT @result = Encryptbykey(@key, 'OMG You Killed Kenny');
SELECT CONVERT(VARCHAR, Decryptbykey(@result))

CLOSE symmetric KEY chavesimetrica01
```

- Algoritmos de chave simétricos podem ser determinístico ou aleatório (= mais seguro)
- É obrigatório sempre abrir e depois fechar a chave para utilizar as funções citadas.
- Preciso converter para o tipo original dado decriptografado (neste caso, VARCHAR).

Criptografia - Funções internas

```
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
--=X=-- CHAVE ASSIMÉTRICA
--=X=-- --=X=-- --=X=-- --=X=-- --=X=-- --=X=--
CREATE asymmetric KEY chaveassimetrica001 WITH algorithm = rsa_2048 encryption
BY password = N'!@@QW#E#$R%dreud76';
go

DECLARE @key_ID INT = (SELECT Asymkey_id('ChaveAssimetrica001'))
DECLARE @result VARBINARY(max)

SELECT @result = Encryptbyasymkey(@key_ID, 'OMG You Killed Kenny')

SELECT CONVERT(VARCHAR, Decryptbyasymkey(@key_ID, @result, N'!@@QW#E#$R%dreud76' ) )
```

- Algoritmos assimétricos podem ser determinísticos ou Aleatórios.
- Não precisam de comandos para abrir ou fechar (em comparação com a anterior).
- São mais seguros [porém mais lentos] que os simétricos, ainda assim, são os mais indicados para criptografia de dados em repouso.

Correção - AC2

```
--Criando a função de criptografia 2-way
```

Recebe uma string e devolve o respectivo valor criptografado em VARBINARY utilizando um algoritmo de 2 vias assimétrico de criptografia.

```
CREATE FUNCTION Fn_encrypt (@conteudo VARCHAR(max))
returns VARBINARY(max)
AS
  BEGIN
      DECLARE @key_ID INT = (SELECT Asymkey_id('ChaveAssimetrica001'))
      DECLARE @result VARBINARY(max)

      SELECT @result = Encryptbyasymkey(@key_ID, @conteudo)

      RETURN @result
  END
go

--Usando a Função de criptografia
SELECT dbo.Fn_encrypt('oi')
```

Correção - AC2

```
--Criando função de decriptografia
```

Recebe um valor já criptografado e devolve o respectivo valor descriptografado já convertido em VARCHAR

```
CREATE FUNCTION Fn_decrypt (@valorCriptografado VARBINARY(max))
returns VARCHAR(max)
AS
  BEGIN
      DECLARE @key_ID INT = (SELECT Asymkey_id('ChaveAssimetrica001'))
      DECLARE @result VARBINARY(max)

      SELECT @result = Decryptbyasymkey(@key_ID, @valorCriptografado,
                       N'!@@QW#E#$R%dreud76')

      RETURN CONVERT(VARCHAR(max), @result)
  END

go
--Usando a função de decriptografia
SELECT dbo.Fn_decrypt(dbo.Fn_encrypt('oi'))
```

Correção - AC2

```
--Criando função de criptografia 1-way HASH
```

Recebe uma string e devolve o respectivo valor criptografado em VARBINARY utilizando um algoritmo de HASH com a utilização de um SALT (por segurança).

```
CREATE FUNCTION Fn_hash (@conteudo VARCHAR(max))
returns VARBINARY(max)
AS
  BEGIN
      DECLARE @result VARBINARY(max)
      DECLARE @SALT VARCHAR(60) = 'salzinho'
      DECLARE @Conteudo_com_sal VARCHAR(max) = @conteudo + @SALT

      SELECT @result = Hashbytes('SHA2_512', @Conteudo_com_sal)

      RETURN @result
  END
go
--Usando a função de criptografia de HASH
SELECT dbo.Fn_hash('oi')
```

Correção - AC2

```
--Criando a tabela TBL_CTRL_ACESSO
     CREATE TABLE tbl_ctrl_acesso
       (  [login]      VARCHAR(60) NOT NULL,
          [senha]      VARBINARY(max) NOT NULL,
          [dica_senha] VARBINARY(max) NULL,
          CONSTRAINT pk_ctrl_acesso PRIMARY KEY ( [login] )
       )
--Inserindo valores nas tabelas para testes:
INSERT INTO tbl_ctrl_acesso( [login], [senha], [dica_senha] )
VALUES      ( 'José', dbo.Fn_hash('senha'),dbo.Fn_encrypt('aquela lá') )
go
--Testando valores brutos inseridos na tabela
SELECT * FROM   tbl_ctrl_acesso
go
--Testando valores decriptografados lidos da tabela
SELECT [login], [senha],
       CONVERT(VARCHAR, dbo.Fn_decrypt([dica_senha])) AS [dica_senha]
FROM   tbl_ctrl_acesso
go
```

Correção - AC2

```
--Criando a Procedure de login:
```

Que, recebe um login e senha (ambos varchar) e devolve 1 se ele foi autenticado, ou seja, se aquele usuário foi cadastrado com aquela senha, e 0 caso contrário.

```
CREATE PROCEDURE Pr_login(@login       VARCHAR(60),
                          @senha       VARCHAR(60),
                          @Autenticado BIT output)
AS
  BEGIN
      SET @Autenticado = 0 --Por padrão ele é 0, só vira 1 se for validado.
      SELECT @Autenticado = 1
      FROM   tbl_ctrl_acesso
      WHERE  [login] = @login
             AND [senha] = dbo.Fn_hash(@senha)
      RETURN @Autenticado
  END
go

EXEC Pr_login  @login,   @senha,   @result output
```

Exemplo de utilização:

```
--testando procedure de login
```

Correção - AC2

```
--Exemplos de utilização:
EXEC Pr_login  @login,   @senha,   @result output

--testando procedure de login
DECLARE @result BIT
--autenticado
EXEC Pr_login   'josé',   'senha',   @result output
SELECT CASE WHEN @result = 1 THEN 'Autenticado' ELSE 'Não autenticado' END

--não autenticado
EXEC Pr_login   'josé',   'senha errada',   @result output
SELECT CASE WHEN @result = 1 THEN 'Autenticado' ELSE 'Não autenticado' END
go
```

Correção - AC2

```
-- CRIANDO PROCEDURE PARA ESQUECI SENHA
```

Que, recebe um login (varchar) e devolve a dica da senha decriptografada, cadastrada para aquele login.

```
CREATE PROCEDURE Pr_esqueci_senha(@login      VARCHAR(60),
                                  @dica_senha VARCHAR(60) output)
AS
  BEGIN
      SELECT @dica_senha = dbo.Fn_decrypt(dica_senha)
      FROM   tbl_ctrl_acesso
      WHERE  [login] = @login
  END

go

EXEC Pr_esqueci_senha   @login,   @result output
```

Exemplo de utilização:

```
--Testando a procedure esqueci senha
DECLARE @result VARCHAR(60)
```

Correção - AC2

```
-- Exemplos de utilização:
EXEC Pr_esqueci_senha   @login,   @result output

--Testando a procedure esqueci senha
DECLARE @result VARCHAR(60)

EXEC Pr_esqueci_senha   'josé',   @result output

SELECT 'Sua dica da senha é: "' + @result + '"'
go
```

Criptografia ‘transparente’

Objetivo:

Interceptar os "selects" da tabela TBL_CTRL_ACESSO em uma visão, que já devolve o dado decriptografado. Esta visão seria a apresentação da tabela (que tem os dados criptografados).

Interceptar nesta visão, usando uma trigger de inserção, os "inserts" que virão com os dados abertas (ou descriptografados) e inseri-los, usando as funções criadas na AC2, na tabela TBL_CTRL_ACESSO, os dados já criptografados....

Desta forma:

TBL_CTRL_ACESSO - terá os dados criptografados

LOGIN	SENHA	DICA_SENHA
Novo Login	0x2A565BD63EA08ADD3047D749A6B13D01C82ECDDFB7BA2D2...	0x1D43C0DB42CBD589D7A14FA26F3A3A54E2748861DDEB27...

vw_TBL_CTRL_ACESSO - terá dados (no caso a dica senha) decriptografados (p/ insert e select)

LOGIN	SENHA	DICA_SENHA
Novo Login	*V[Ö> ŠÝ0G×l¦±=È.ß·º–¤ÍÂÏp	Nova Dica Senha

Criptografia ‘transparente’

```
--Criando a visão, que servirá como fachada para a tabela.
CREATE OR ALTER VIEW vw_tbl_ctrl_acesso AS
SELECT [login]
       --A coluna senha deve ser tipada como a 'fachada'.
       , Cast(senha AS VARCHAR) AS SENHA
       --A dica da senha já será varchar ( decriptografada )
       , dbo.Fn_decrypt(dica_senha) AS DICA_SENHA
FROM   tbl_ctrl_acesso;go

--Testando a visão
SELECT * FROM   vw_tbl_ctrl_acessogo
```

Criptografia 'transparente'

```
--Criando trigger para captura de inserção.
CREATE TRIGGER trg_II_tbl_ctrl_acesso ON vw_tbl_ctrl_acesso
instead OF INSERT
AS
  BEGIN
      INSERT INTO tbl_ctrl_acesso([login],senha,dica_senha)
      SELECT [login],
             dbo.Fn_hash(senha),
             dbo.Fn_encrypt(dica_senha)
      FROM   inserted
  END
--Testando a inserção na visão ( fachada da tabela )
INSERT INTO vw_tbl_ctrl_acesso
VALUES      ('Novo Login','Nova Senha','Nova Dica Senha');
go
-- Conferindo a diferença entre a visão e a tabela
SELECT * FROM   vw_tbl_ctrl_acesso
SELECT * FROM   tbl_ctrl_acesso
```

Always Encrypted

Da mesma maneira que nossa versão caseira de criptografia 'transparente' a técnica de always encrypted fornece uma versão muito outra maneira para o uso de criptografia quando aplicada em dados dentro de colunas.

- Ela sempre utiliza o algoritmo: AEAD_AES_256_CBC_HMAC_SHA_256

Mais detalhes em:

Always Encrypted cryptography - SQL Server | Microsoft Docs

draft-mcgrew-aead-aes-cbc-hmac-sha2-05 - Authenticated Encryption with AES-CBC and HMAC-SHA (ietf.org)

- Podem ser determinísticos ou aleatórios.
- Podem ser utilizadas com chaves (criptografadas com outros algoritmos, se desejado).
- Pode ser realizado utilizando a interface SSMS ou via código SQL
- Inviabiliza comandos tradicionais como INSERTs nas colunas criptografadas, porém, ainda permitem que sejam realizados via linguagem de programação (ou chamadas parametrizadas, mesmo dentro do SSMS).

Mais detalhes:

Always Encrypted - SQL Server | Microsoft Docs

SQL Server Encryption: Always Encrypted - Simple Talk (red-gate.com)

Exploring SQL Server 2016 Always Encrypted – Part 4 – Encrypting Existing Data — DatabaseJournal.com

Getting Started with Always Encrypted Part 2 | Microsoft Docs

Always Encrypted

```
/*
Testando [parcialmente] a técnica always encrypted
*/
USE [Cripto]
go
CREATE column master KEY alwaysencrypted_masterkey
WITH ( key_store_provider_name = 'MSSQL_CERTIFICATE_STORE'
     , key_path = 'Current User/Personal/f2260f28d909d21c642a3d8e0b45a830e79a1420'
);
-------------------------------------------
CREATE COLUMN encryption KEY alwaysencrypted_encryptionkey
WITH VALUES ( column_master_key = alwaysencrypted_masterkey
              , algorithm = 'RSA_OAEP'
              , encrypted_value = 0x01700000016C006F00630.... );
-------------------------------------------
go
```

Always Encrypted

```
CREATE TABLE tbl_ctrl_acesso_alwaysencrypted
 (   [LOGIN]       VARCHAR(60) NOT NULL ,
     [SENHA]       VARCHAR(255)
          COLLATE latin1_general_bin2 encrypted
          WITH ( encryption_type = deterministic
               , algorithm = 'AEAD_AES_256_CBC_HMAC_SHA_256'
               , column_encryption_key = alwaysencrypted_encryptionkey ) ,
     [DICA_SENHA] varchar(255)
          COLLATE latin1_general_bin2 encrypted
          WITH ( encryption_type = randomized
               , algorithm = 'AEAD_AES_256_CBC_HMAC_SHA_256'
               , column_encryption_key = alwaysencrypted_encryptionkey ) ,
     CONSTRAINT pk_ctrl_acesso_alwaysencrypted PRIMARY KEY ( [LOGIN] )
     )
     go
```

Always Encrypted

```
--Mas as facilidades acabam por ai...
-- O seguinte comando já dá erro:
INSERT INTO tbl_ctrl_acesso_alwaysencrypted
VALUES      ('Novo Login',
             'Nova Senha',
             'Nova Dica Senha');
go
Mensagem 206, Nível 16, Estado 2, Linha 87
Conflito no tipo de operando: varchar é incompatível com varchar(8000) encrypted with
(encryption_type = 'DETERMINISTIC', encryption_algorithm_name =
'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name =
'AlwaysEncrypted_EncryptionKey', column_encryption_key_database_name = 'Cripto')
collation_name = 'Latin1_General_CI_AS'
```

Esta técnica não permite mais o uso de comandos simples de SQL para manipulação das informações...

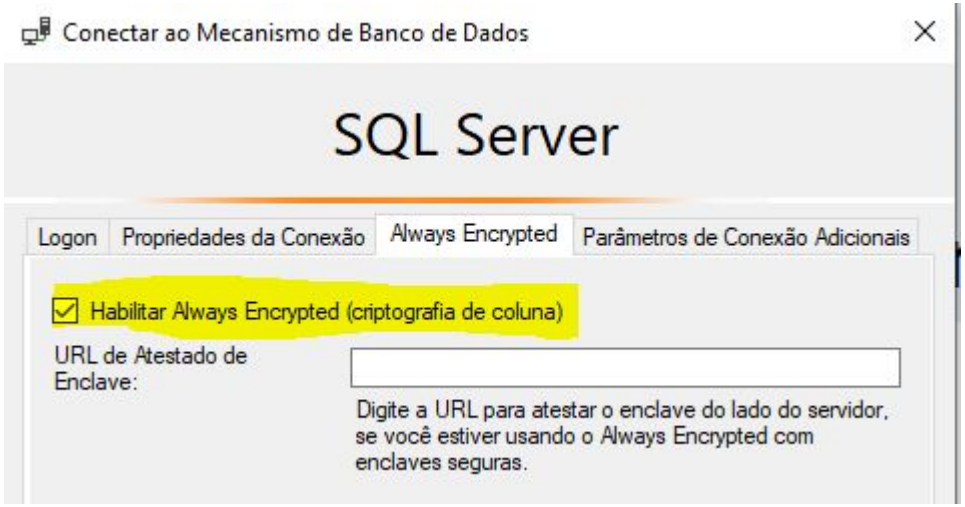
Para entender as dificuldades de utilização de comandos SQL em colunas utilizando o always encrypted, consultem:

Parameterization for Always Encrypted - Using SSMS to Insert into, Update and Filter by Encrypted Columns | Microsoft Docs

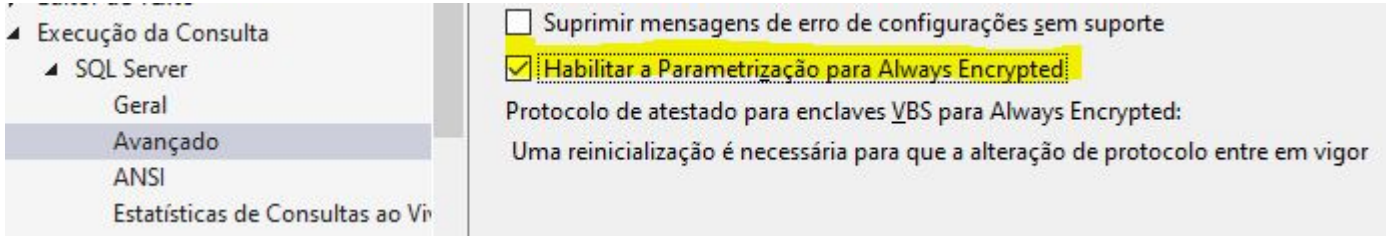
Getting Started with Always Encrypted Part 2 | Microsoft Docs

Always Encrypted

-- NA CONEXÃO: HABILITAR USO DE CRIPTOGRAFIA DE COLUNAS



-- NAS CONFIGURAÇÕES DO SSMS, HABILITAR O USO DE PARAMETRIZAÇÃO



Always Encrypted

```
--Agora todo SQL deve só ser realizado com parâmetros
DECLARE @login      VARCHAR(60) = 'Novo Login',
        @SENHA      VARCHAR(255) = 'Nova Senha',
        @DICA_SENHA VARCHAR(255) = 'Nova Dica Senha'

INSERT INTO tbl_ctrl_acesso_alwaysencrypted
VALUES      (@login,
             @SENHA,
             @DICA_SENHA);

go
```

OU deve-se utilizar de bibliotecas ou métodos específicos nas linguagens de programação para a devida utilização desta técnica.

TDE - Transparent Data Encryption

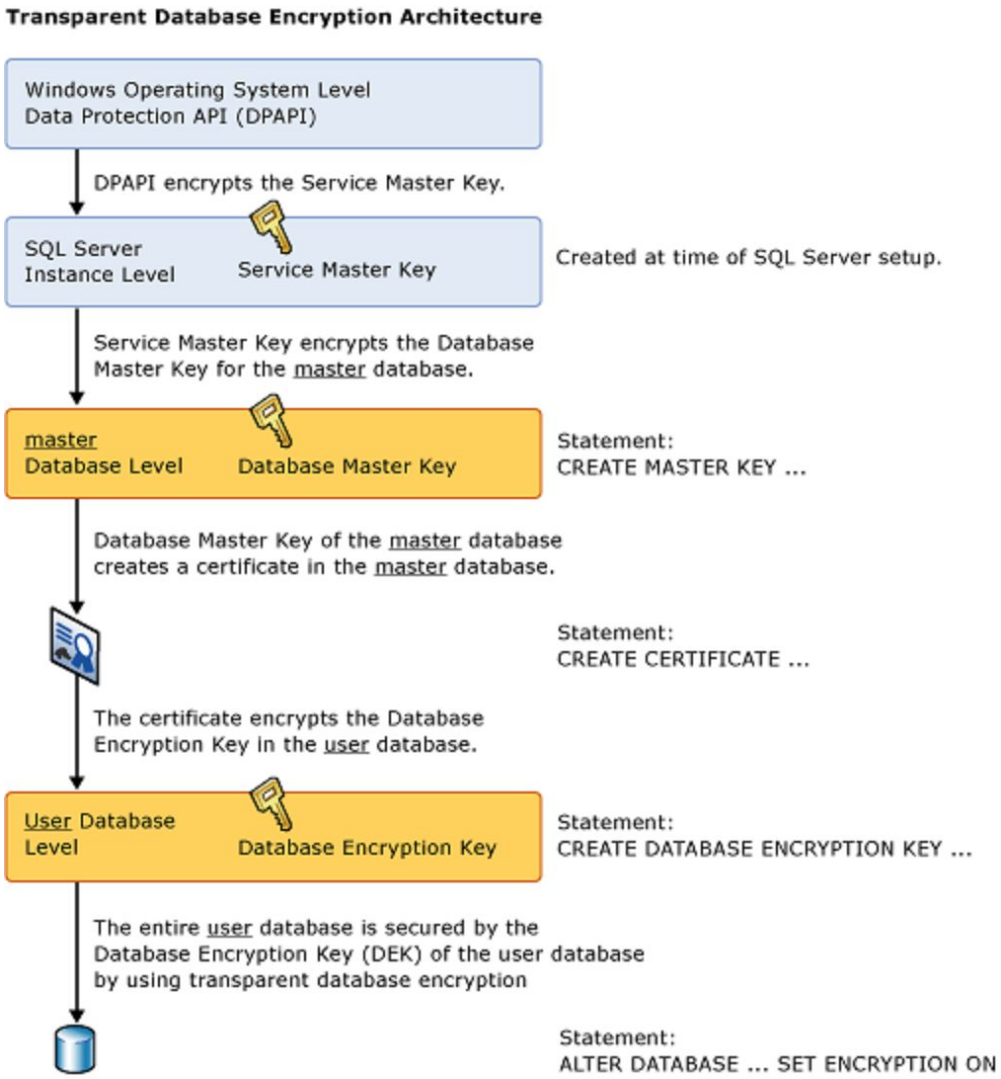
Das técnicas aprendidas até o momento, esta é a que realmente leva transparência a sério, inclusive no nome, porém:

- Esta é uma técnica em nível de banco de dados, ou seja, o banco inteiro é criptografado. Isso significa que, quem tem acesso ao banco terá acesso aos dados limitado apenas pelo controle de acesso aos schemas, tabelas, etc.
- O que realmente fica 'protegido' nesta técnica são os arquivo físicos .MDF, .NDF, .LDF do banco de dados, assim como todos os backups realizados destes.
- Praticamente TODOS os comandos SQL continuam funcionando de forma transparente para o usuário.

Mais detalhes:

TDE (Transparent Data Encryption) - SQL Server | Microsoft Docs

TDE - Transparent Data Encryption



Fonte: [TDE (Transparent Data Encryption) - SQL Server | Microsoft Docs]

TDE - Transparent Data Encryption

```
USE master;
go
--Crio uma chave mestra ( para o servidor )
CREATE master KEY encryption BY password = N'!@@QW#E#$R%dreud76';
go
--Crio um certificado ou obtenho um certificado assinado
CREATE certificate meucertificadoparatde WITH subject = 'Meu Certificado para TDE';
go
--Agora dentro do banco
USE cripto;
go
--Crio a chave de criptografia de banco de dados
-- protegido pelo certificado gerado
CREATE DATABASE encryption KEY WITH algorithm = aes_128 encryption
     BY server certificate meucertificadoparatde;
go
-- Ligo a criptografia
ALTER DATABASE cripto SET encryption ON;
go
```

Obrigado

Segurança de dados

Aula 07 – Criptografía - parte 2

- **Gustavo Bianchi Maia**

gustavo.maia@faculdadeimpacta.com.br