UNIVERSIDADE FEDERAL
DE SANTA CATARINA

ine
CTC • UFSC
Informática e Estatística

INE 5680
Segurança da Informação e de Redes

Criptografia – Derivação de chaves

Profa: Carla Merkle Westphall
carla.merkle.westphall@ufsc.br

Derivação de chaves

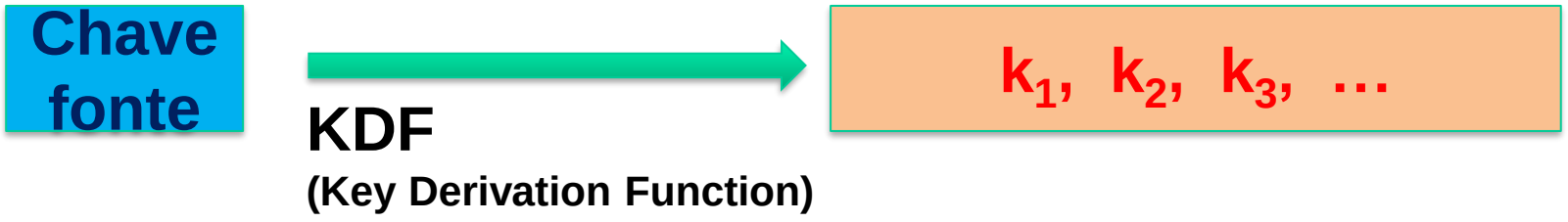
Objetivo: derivar várias chaves a partir de UMA chave fonte (*source key* – SK)

Cenário típico: uma única chave é derivada de:

- **Gerador de números aleatórios em hardware**
- **Um protocolo de trocas de chaves**

Várias chaves são necessárias para a segurança de uma sessão:

- **Chaves unidirecionais; múltiplas chaves para CBC baseados em nonces.**



Key Derivation Function (KDF)

Supondo SK é uniforme em K, define-se KDF como:

KDF(SK, CTX, L) :=

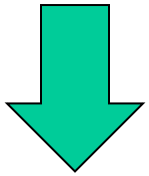
F(SK, (**CTX II 0**)) II F(SK, (**CTX II 1**)) II ⋯ II F(SK, (**CTX II L**))

- **SK: source key – chave fonte**
- **CTX: uma string que identifica unicamente uma aplicação (servidor web, ssh, ipsec, …). Mesmo que duas aplicações tenham a mesma SK, elas terão chaves diferentes!**
- **L: tamanho**

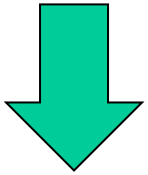
Se SK não é uniforme?

Normalmente chaves SK não são uniformes quando:

- **distribuídas em protocolos de trocas de chave**
- **obtidas a partir de HRNG (*hardware random number generator*)**



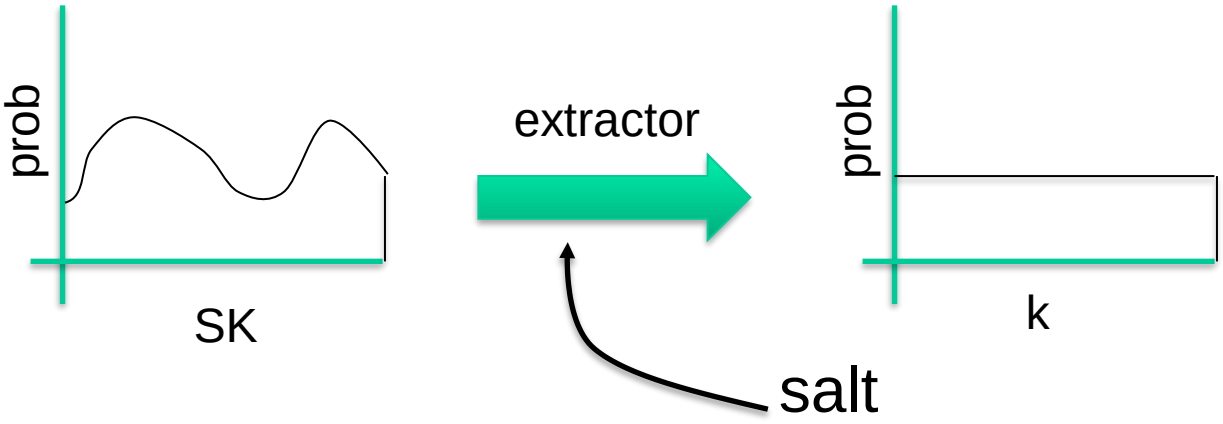
Nunca usar essas chaves obtidas diretamente!



Usar o paradigma Extrair e Expandir para obter chaves adequadas para as sessões.

Paradigma Extrair e Expandir

Passo 1: extrair uma chave k pseudo-aleatória da chave fonte SK



salt: um string ALEATÓRIO fixo, público (não secreto)

Passo 2: expandir k usando esta chave k em uma função PRF (*pseudo random function*)

HKDF: uma função KDF feita com HMAC

Implementa o paradigma extrair e expandir:

http://tools.ietf.org/html/rfc5869

- **extrair: usar** $k \leftarrow HMAC(salt, SK)$
- **Então expandir usando HMAC como uma função PRF com chave k**

Password-Based KDF (PBKDF)

Senhas não devem ser usadas diretamente como chaves porque:

- Não tem entropia suficiente
- Não tem propriedades suficientes de aleatoriedade

PBKDF defende contra falta de entropia com :

salt (contra ataque do dicionário) e

função hash lenta (repetida MUITAS vezes)

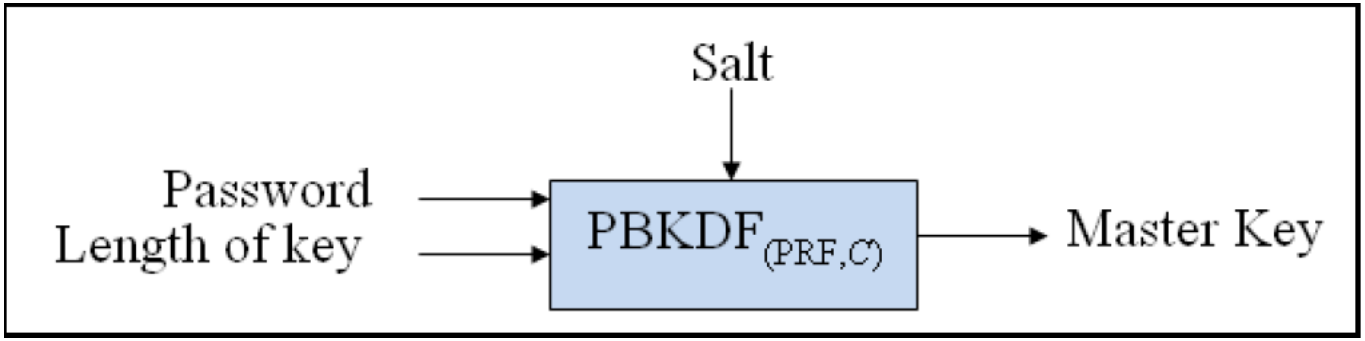
PBKDF2 (Password-Based Key Derivation Function 2)

- Abordagem padrão PKCS#5 (Public-Key Cryptography Standards do Laboratório RSA)
- Ano 2000 - RFC2898 - http://tools.ietf.org/html/rfc2898
- Publicação Especial NIST 800-132:
  - http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

PBKDF2

$H^{(c)}$(password ll salt): repetir a função hash "c" vezes

k = PBKDF2(PRF, Password, Salt, c, dkLen)



- *PRF* é a função pseudo-aleatória (HMAC com chave)
- *password* é a senha fornecida
- *salt* deve ser ALEATÓRIO e ser gerado com um gerador de bits aleatórios
  - Usar salt = CTX ll valor aleatório. O tamanho do salt é importante.
- *c (count)* número de vezes que a função PRF é executada
  - Mínimo 1000 (100.000?). Para sistemas críticos 10.000.000 pode ser apropriado.
- *dkLen* é o comprimento da chave derivada k (no mínimo 112)

Sistemas que usam PBKDF2

- ✓ **WPA2 - IEEE 802.11i: Pre-Shared Key (PSK)**
- ❑ **Mapeamento Senha-para-Chave**
  - ❑ Usa PKCS #5 v2.0 PBKDF2 para gerar uma chave PSK de 256-bit a partir de uma senha ASCII
  - ❑ PMK=PSK = PBKDF2 (Password, SSID, SSIDlength, 4096, 256)
    - ❑ Salt = SSID, assim diferentes PSKs para diferentes SSIDs
    - ❑ 4096 é o número de hashes usados no processo
- ✓ **Windows Data Protection API, WinZip, LastPass, Cisco IOS, …**
- ✓ **TrueCrypt, EncFS (Linux, MacOS), …**

WPA3-Enterprise (https://www.wi-fi.org/discover-wi-fi/security)

- Criptografia Autenticada: AES com chaves de 256 bits no modo GCM
- Derivação de chaves: HMAC-SHA384
- Estabelecimento de chaves e autenticação: troca ECDH (Elliptic Curve Diffie-Hellman) e assinatura com algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm) usando curvas elípticas de 384 bits. Uso do RSA com chaves >= 3072 bits

Outros opções importantes de KDF

- **Bcrypt**
  - Proposto em 1999, projeto do OpenBSD, Niels Provos e David Mazières
  - Derivado do Blowfish ao invés de repetir função hash
  - Em Java: http://www.mindrot.org/projects/jBCrypt/
- **Scrypt**
  - Colin Percival
  - baseado em funções de hardware
  - http://www.tarsnap.com/scrypt/scrypt.pdf
  - Em Java: https://github.com/wg/scrypt
- **Novos KDF: Password hashing competition**
  - https://password-hashing.net/
  - Argon2
  - Outros finalistas: Catena, Lyra2 (Brasil), Makwa, yescrypt

Histórico importante sobre senhas

- Comparações interessantes entre métodos ao longo do tempo

  http://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/Passwords12-The-Future-Of-Hashing.pdf