Criptografia e Segurança na Informática

Capítulos 3 e 4

Pedro A. D. Rezende

Ciência da Computação

Universidade de Brasília

3: Protocolos Importantes

- **Necessidade de protocolos para a utilidade da Criptografia**

Criptografia não cria proteção do nada. Suas primitivas são capazes apenas de processar certos modos de confiança, por meio de mecanismos criptográficos que assim protegem certos valores, em situações onde tais modos ocorrem:

Autorização *transforma presunção de sigilo e integridade externos (de senha, token ou chave) e integridade de canais internos, em identificação de um agente cadastrado, para aplicação de permissões de acesso*.

Cifragem *transforma presunção de sigilo e integridade de chave(s) em presunção do sigilo de mensagem cifrada, durante a transmissão desta*

Autenticação *transforma presunção de sigilo (de uma chave secreta ou privada) e de integridade (destas ou de uma chave pública) em validação de origem e integridade, para transmissões do conteúdo autenticado.*

Certificação *transforma presunção de sigilo e/ou integridade (de chaves assimétricas) em autenticação recursiva, para validação objetiva.*

Um protocolo criptográfico processa os modos de confiança que presume, se presentes em situações comunicativas que demandam proteção, fornecendo-a se seu propósito for adequado à demanda, e se sua implementação estiver sadia.

- **Protocolos "básicos" - Autenticação para controles fundamentais**

O desenho de um protocolo criptográfico básico presume condições mínimas de confiança "como entrada". Esse mínimo depende da arquitetura da plataforma onde se propõe buscar uma proteção básica. Tais protocolos podem então ser classificados conforme o tipo de arquitetura a que servem *adequadamente*.

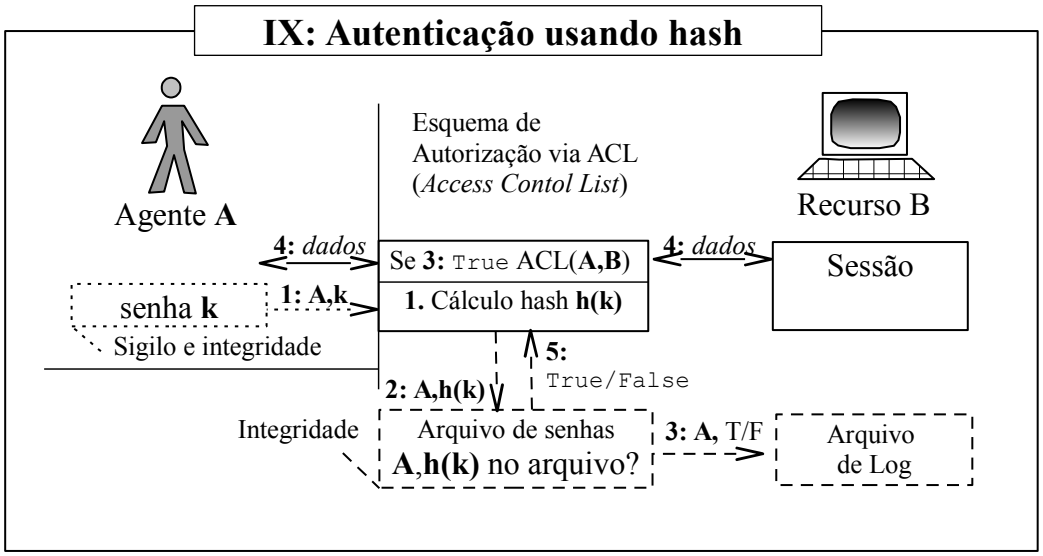
Login: autenticação para controle de acessos *em sistema fechado;*

Distribuição de Chaves: autenticação p/ controle de seções *em rede fechada;*

PKI ou **ICP:** autenticação para controle de identidades *em rede aberta.*

Login

- **Autenticação para controle de acesso por senha:**



Premissas e limitações dos protocolos de Login -

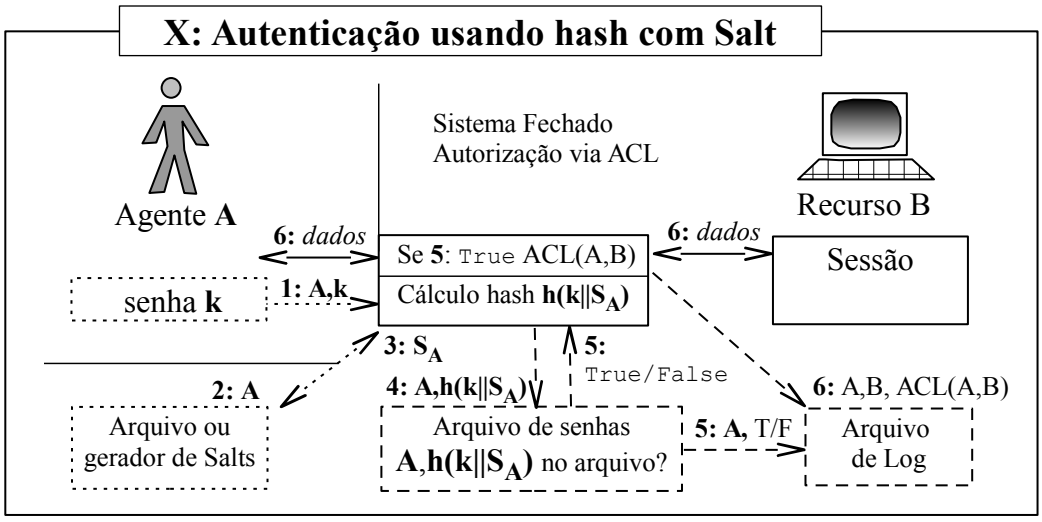
a - Passos 1, 2 e 3 e o arquivo de senhas hasheadas devem ser protegidos (ver Apêndice B). A função de hash deve ser livre de colisão (o Login foi que deu origem à demanda por funções de *hash criptográfico,* durante o projeto dos primeiros sistemas operacionais multiusuários)

b - O passo 1 é o elo mais fraco de qualquer sistema de controle de acesso, pois envolve premissas de confiança de um canal de comunicação que é externo ao sistema digital em foco.

c - Personificação pode advir do vazamento do arquivo de senhas hasheadas [...A,h(k)...], pois **K** tem distribuição de probabilidade heterogênea. Ataques via texto pleno escolhido nesse arquivo são chamados **ataques de dicionário**

d - A estado da arte em ataques de dicionário para vazamento de senhas hasheadas pode ser pesquisado com o termo "password cracking". As técnicas para esse tipo de ataque e sua evolução estão registradas, por ex., na Wikipedia: https://en.wikipedia.org/wiki/Password_cracking

- **Dificultando o ataque de dicionário usando salt:**



Premissas e limitações do protocolo X:

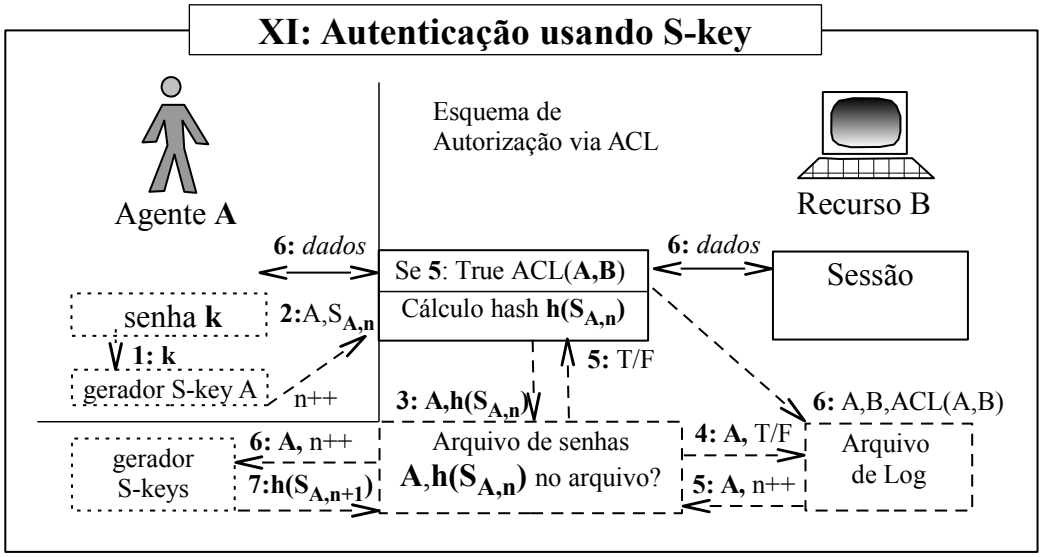
a - O *salt* é uma sequência randômica de uso semelhante ao *nounce* (ver XII) ou ao segredo no esquema MAC (pag. 30). Como entrada adicional para hashear senhas, individualizada para um sistema e para um par usuário-senha, serve para aumentar a entropia dos hashes e para limitar a uma única instância o uso direto da busca de colisão para personificação.

b - Ataques de dicionário têm assim sua eficácia reduzida pela "localização" da relação $\mathbf{A},\mathbf{S_A}$**:** Se uma senha **k** estiver "colidindo" entre usuários, cadastrada para A e A' em um ou mais sistemas que hasheiam senhas com o mesmo padrão de função **h( )**, tal colisão não será detectável por mera inspeção dos arquivos de senhas hasehadas, já que $\mathbf{h(k||S_A)} \neq \mathbf{h(k||S_{A'})}$.

c -Adicionalmente, quando o hash da senha é um campo da ACL, em arquiteturas como a posix, armazena-se o *salt* junto com o hash em arquivo de acesso restrito a processos que precisam autenticar usuários no login (na arquitetura posix, no arquivo chamado *shadow password)*.

d -O salt pode ser único para cada senha ($\mathbf{S_A}$) e também para cada sessão ($\mathbf{S_{A,n}}$), o que requer sincronização das tabelas de *salt* e de hash de senhas. O prórpio salt $\mathbf{S_{A,n}}$ pode assim se tornar a senha, chamada *s-key* (chave de sessão), com geração no lado do usuário implementada por hardware dedicado, evitando personificação via replay com escuta em rede (ver XI)

- **Prevenindo personificação por replay em login: S-key**

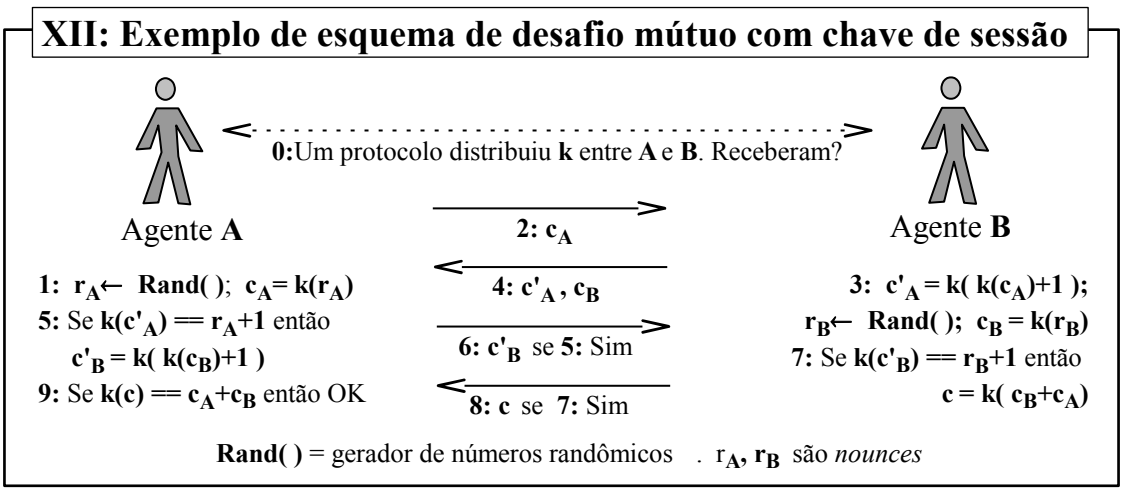


Premissas e limitações do protocolo S-key:

Requer sincronia entre hw dedicado à identificação do agente e o gerador,, da mesma sequência $\mathbf{S_{A,n}}$ no sistema remoto que o identifica. Evita personificação por escuta na autenticação de usuário, mas não outros ataques de replay em rede. Não serve, portanto, para controle de acesso em rede.

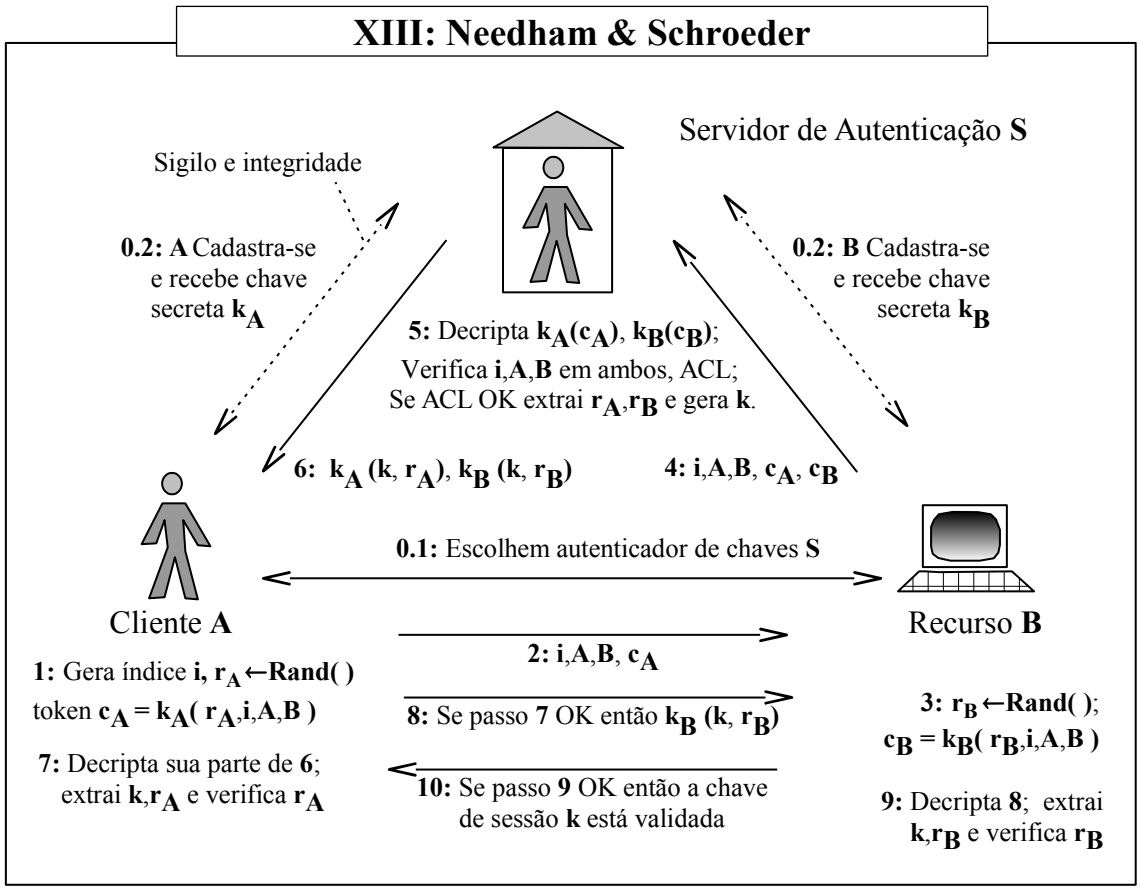
- **Desafio Criptográfico com *Nounce*:**

É um tipo de esquema para fazer uma sequência randômica – ***nounce*** – passar por um circuito sob diferentes cifragens e/ou transformações, com o propósito do originador poder verificar a posse de determinadas chave(s) nos pontos por onde passa. Exemplo:



Autenticação e Distribuição de chaves de seção em rede fechada via cifra simétrica

- **Um protocolo pioneiro:**



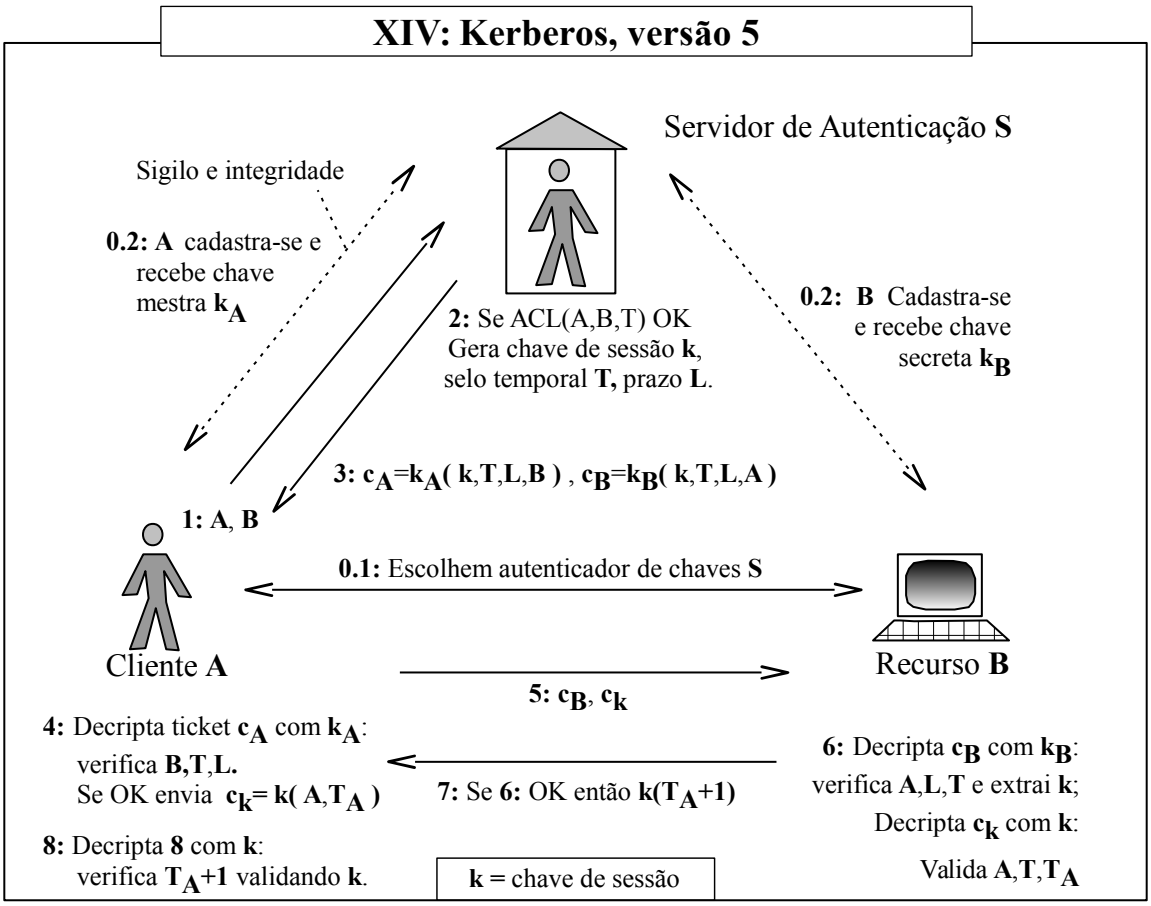
Premissas, propósito e limitações do protocolo XIII:

a- Presume-se que **S** proteja as chaves mestras k_A, k_B, etc. contra vazamento. A informação sobre acessos registrada nos logs presume a lisura de **S** no uso das chaves secretas que **S** gera e compartilha ou distribui (k_A, k_B, **k**, etc) para controlar acesso a recursos em rede.

b- A versão inicial deste protocolo, primeira do gênero (para redes fechadas), não incluía desafio com r_A, r_B, o que possibilitava ataques de replay com **k** comprometidas.

c- A versão acima foi publicada simultaneamente com a descrição de outro protocolo semelhante, conhecido como protocolo de **Otway-Rees**.

- **Protocolo escalável proposto por grupo do MIT: Kerberos**



Premissas, propósito e limitações do protocolo XIV:

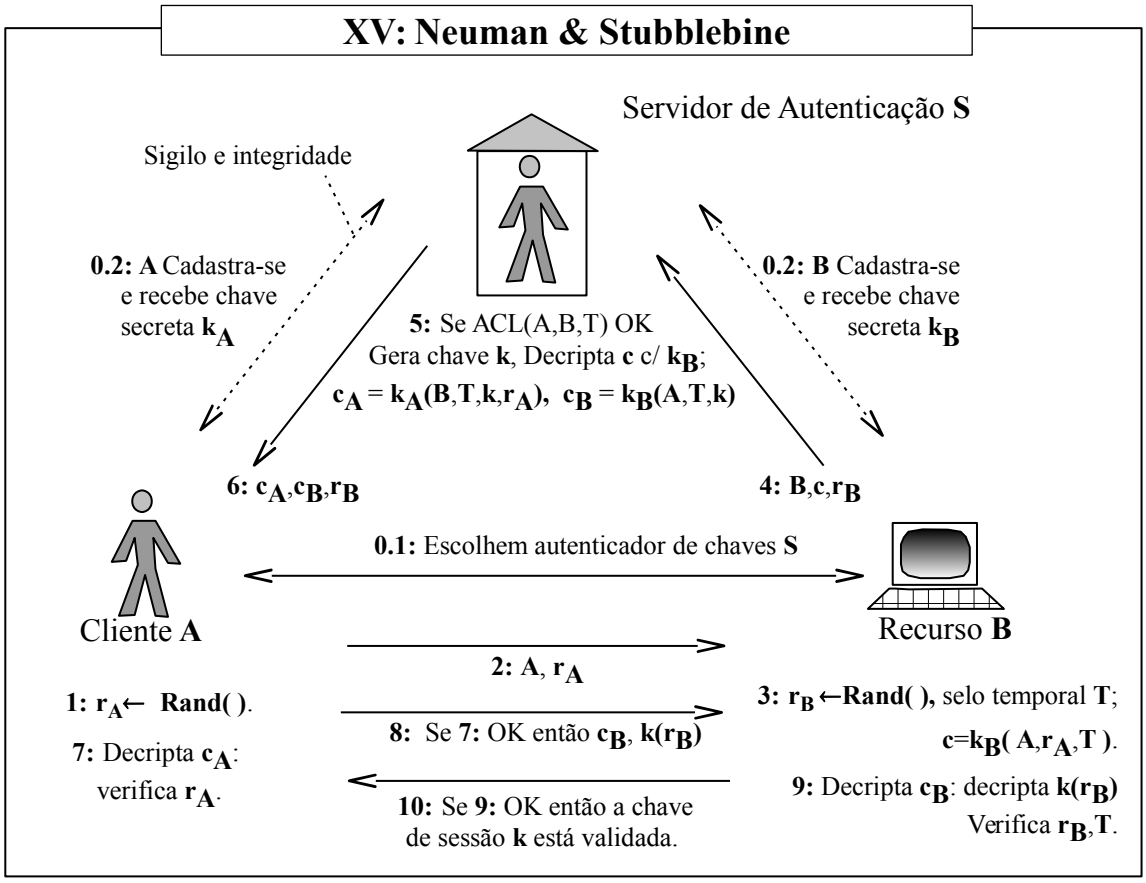
a- Presume-se de **S** o mesmo que em **XIII** (distr. de chaves em rede fechada)

b- Este protocolo não requer de agentes a execução de desafios com *nounces,* mas requer sincronização dos relógios das suas plataformas com o de **S,** e ajuste de latências de transmissões na rede para validações (os selos temporais T, L, T_A, também funcionam, com valores esperados, para efeito de validação de origem dos tickets)

c- c_A transmitido em 5 (denominado *ticket*) contém chave de sessão reutilizável dentro do prazo de validade L, e/ou para geração de novas chaves de sessão pelo cliente nesse prazo. Falhas de sincronismo permitem ataques de negação de serviço (bloqueio) e *replay*.

d- Chaves de sessão geradas pelo cliente são cifradas com a chave anterior, que precisa ser retida em buffer (risco em plataformas multiusuário).

- **Protocolo variante do Kerberos sem sincronização:**



Premissas, propósito e limitações do protocolo XV:

a- Presume-se de **S** o mesmo que em **XIII** (distr. de chaves em rede fechada)

b- O protocolo permite que o *ticket* $\mathbf{c_B}$, transmitido no passo **7** e retrans-mitido no passo **9,** seja utilizado pelos agentes principais (A, B) como um *certificado temporário para mútua autenticação*. Com ele, uma nova chave de sessão pode ser estabelecida sem necessidade do servidor **S**

c- Nova chave de sessão pode ser negociada entre os prinicpais de forma a evitar ataques de replay como segue:

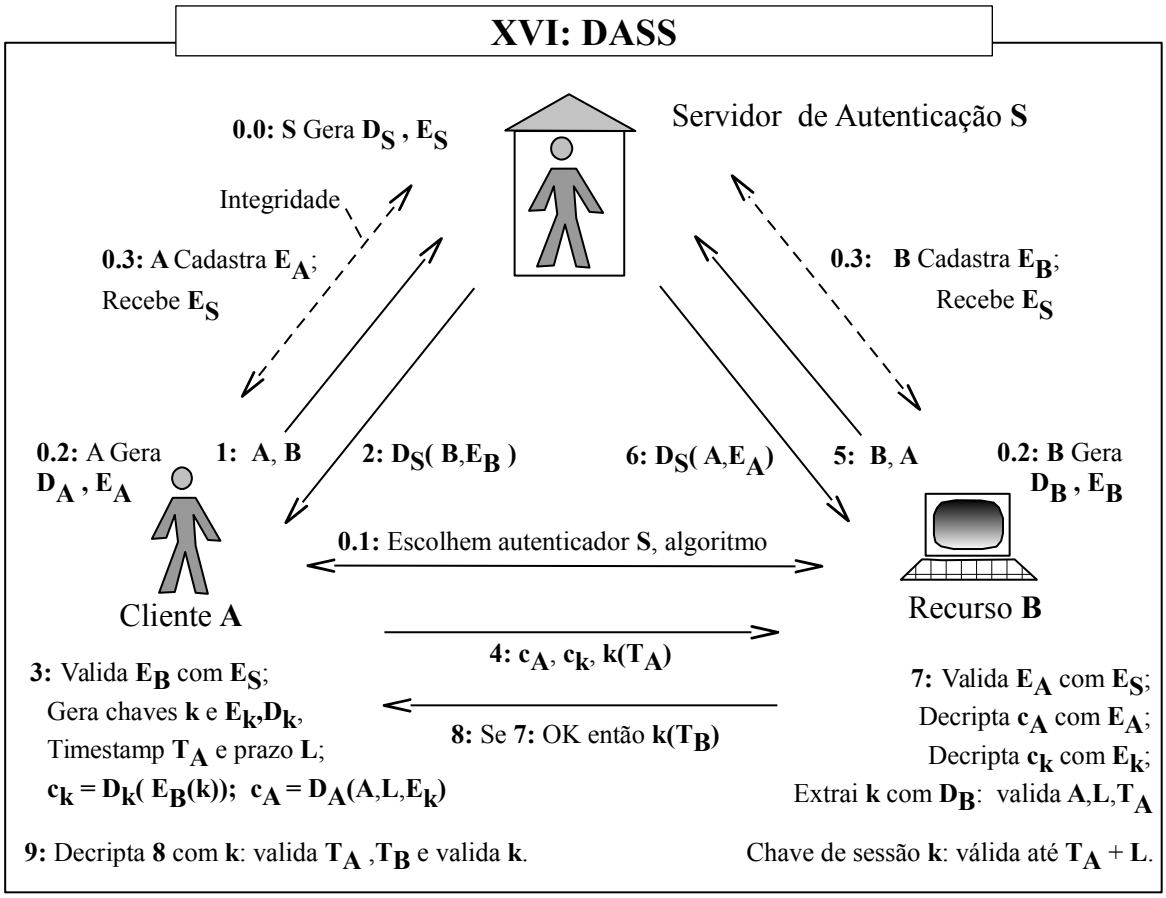
passo **8': A** gera novo *nounce* $\mathbf{r'_A}$ e envia $\mathbf{r'_A}$, $\mathbf{c_B}$ a **B**;

passos **9'**, **10': B** decripta $\mathbf{c_B}$, gera novo *nounce* $\mathbf{r'_B}$ e envia $\mathbf{r'_B}$, $\mathbf{k(r'_A)}$ a **A**;

passo **11: A** valida a posse de **k** por **B** verificando $\mathbf{r'_A}$ e enviando $\mathbf{k(r'_B)}$.

Autenticação e Distribuição de chaves de seção em rede fechada via cifra assimétrica

- **Protocolo proposto pela Digital Equipment Corp: DASS**



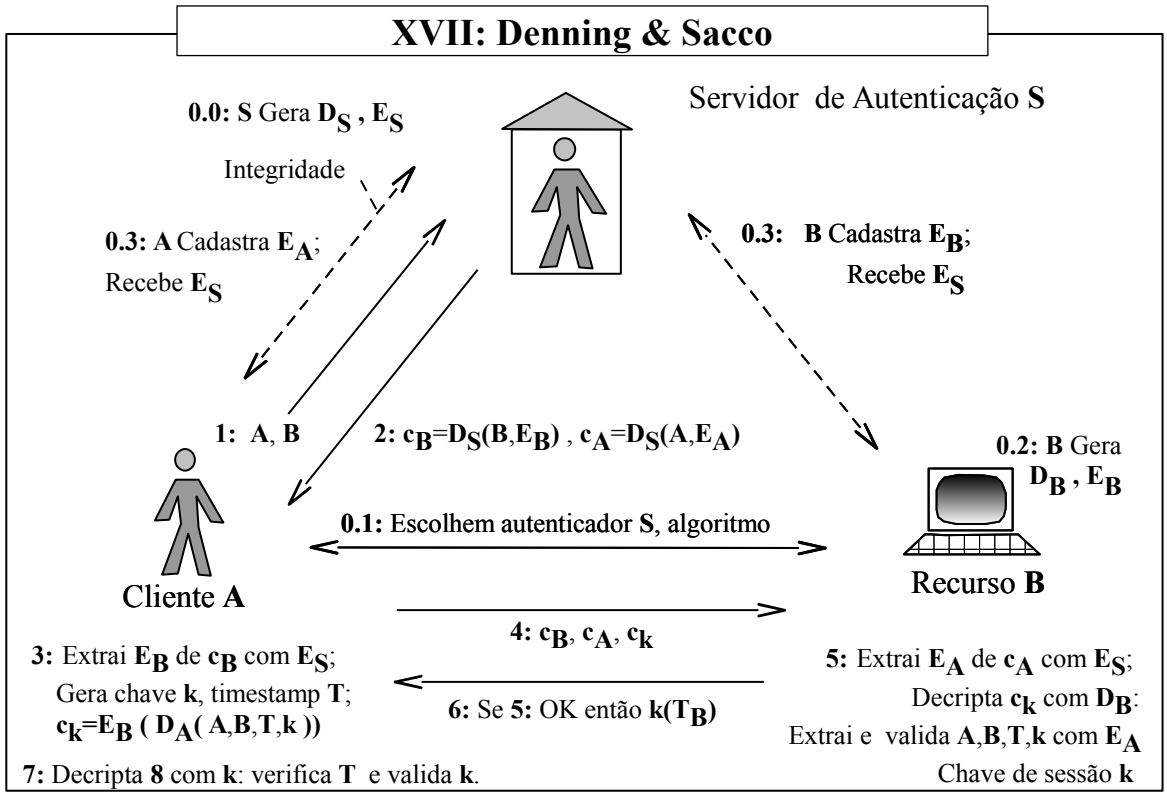
Premissas, propósito e limitações do protocolo XVI:

a- Presume-se que **S** valide as chaves públicas dos principais (E_A, E_B, etc.) contra personificação, e proteja sua chave privada (D_S) contra vazamento. **S** permite controle de acesso distribuído (cada recurso controla acesso a si) via política de certificação de chaves públicas dos agentes da rede.

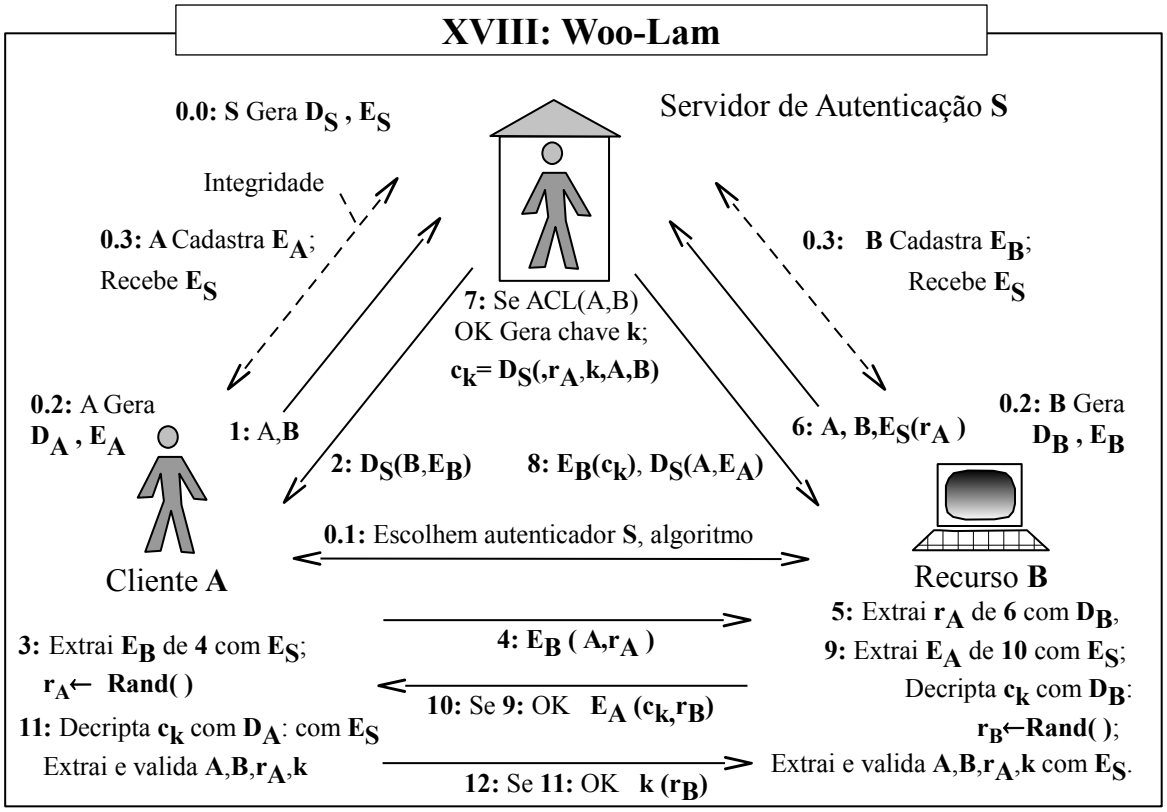
b- O protocolo requer sincronismo entre agentes e permite a distribuição de chaves públicas assinadas por distintos servidores.

- **Outros exemplos de protocolo semelhantes:**

Denning and Sacco: Controle distribuído de acesso por seção



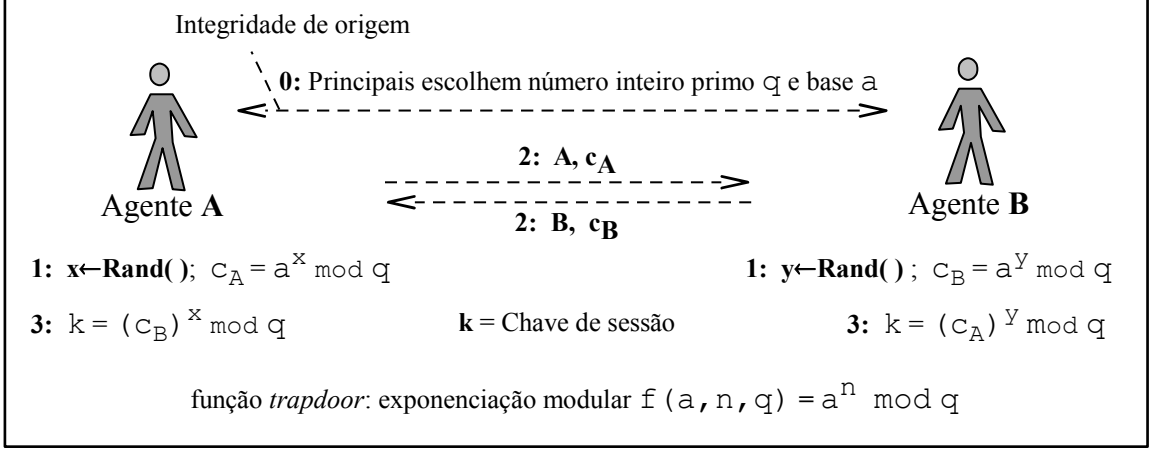
Woo-Lam: Controle centralizado de acesso por seção



Derivação de chaves via função *trapdoor*

- **Primeiro protoalgoritmo assimétrico: Diffie & Hellman** (1976)

XIX: Esquema Diffie & Hellman



Premissas, propósito e limitações do esquema D&H:

a - D&H deriva uma mesma chave de sessão para **A** e **B** no passo 3 porque $f(\_,1,q)$ é homomorfismo entre os anéis $\mathbb{Z}$ e $\mathbb{Z}_q$ (ver Apêndice A-9,11)

$$f(c_B,x,q) = (a^y \bmod q)^x \bmod q = a^{y*x} \bmod q =$$

$$a^{x*y} \bmod q = (a^x \bmod q)^y \bmod q = f(c_A,y,q)$$

b - Para escolha adequada do espaço de chaves **K (**cujo tamanho é dado por $|K| = (q-1)/mdc[$graus de a, x e y em $\mathbb{Z}_q]$), será inviável obter-se **k** a partir de $\mathbf{c_A}$ ou $\mathbf{c_B}$ devido à complexidade do **Problema do Logaritmo Discreto** (PLD: dados a, q, c, encontrar x tal que $a^x \bmod q = c$) (Cap. 4)

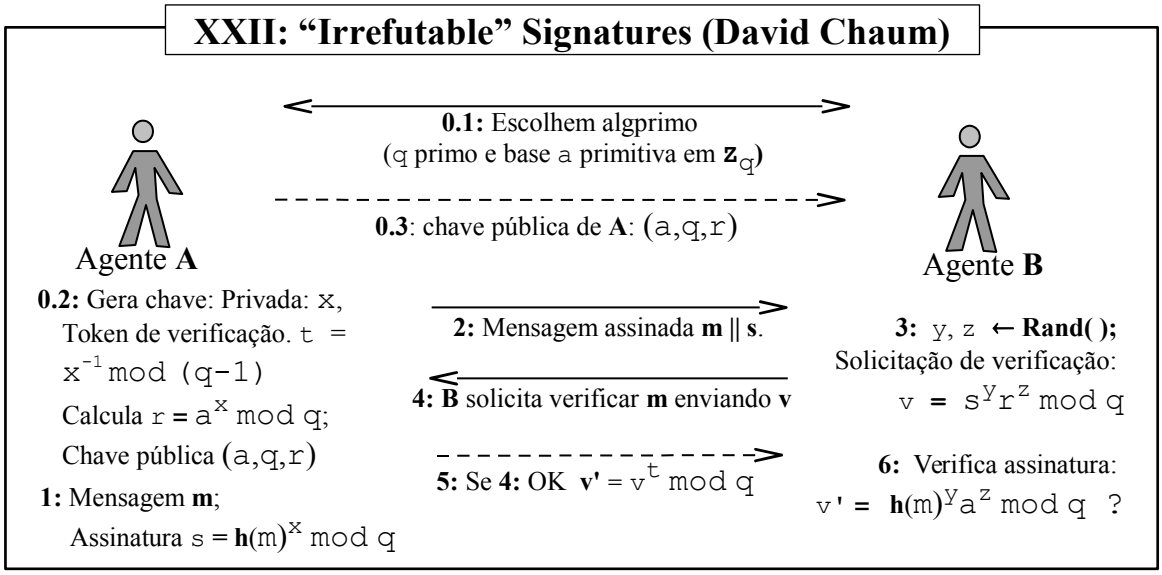
c - Em rede aberta (onde D&H é útil, por não presumir transmissão sigilosa prévia), o protocolo é vulnerável a ataques de espelhamento e *spoofing*, pois presume integridade na identificação prévia precípua dos principais.

d- Esse esquema foi patenteado como protocolo nos EUA, vigente até 1997. Dele foram derivados os algoritmos assimétricos RSA e El Gamal (1978), inversíveis: O RSA, com módulo (q) composto, e o El Gamal, com a base (a) composta – ver Apêndice C –, mas não cobertos pela patente do D&H

Aplicações do PLD e Fatoração (ver Apêndice C)

Variantes do Esquema Meta-ElGamal para algoritmos assimétricos:

- **Assinatura Digital com verificação intermediada pelo signatário**



Premissas, características e limitações do protocolo XXII:

a - Mesmas premissas do esquema de assinatura digital (página 27)

b- O protocolo oferece privacidade na verificação, permitindo ao signatário escolher a quem permitir a prova de sua autoria em documentos, através de sua participação na verificação (enviando **v'**). Apesar de livre de premissa de sigilo compartilhado entre principais, a verificação é subjetiva: **B** não convence terceiros da origem de **m||s**, pois **6** é falsificável e sabotável (um **v'** propositalmente falso pode ser enviado).

c- A refutação em má fé da autoria de assinaturas é não apenas possível, como em IV (página 30), mas também seletivamente preemptiva: **A** pode alegar "perda" do verificador `t` ou da chave privada `x` **(**acarretando impossibilidade de calcular `t`**)** quando lhe convier.

d - Existem variantes de XXII que buscam atenuar os problemas em XXII .**c**, nas quais o signatário é desafiado, em variações ajuizáveis dos passos **3** a **6,** a autenticar `s` ou a provar a forja de `s`. Outras variantes designam a verificação a um agente TTP (semelhantes ao esquema **V**).

- **Assinaturas digitais por procuração:**

  (Mambo Usuda & Ocamoto, SCIS '95)

  Neste protocolo, um procurador assina em nome do principal **A,** sem conhecer a chave privada de **A**. Suas assinaturas são distinguíveis das lavradas por **A**, e tecnicamente irrefutáveis como estas. Da sua verificação pode ser deduzida identificação do procurador e dos poderes a ele delegados para lavrar assinaturas em nome de **A**.

- **Assinaturas inforjáveis:** (Pfitzmann, COMPUSEC '91)

  Neste protocolo ajuizado, uma disputa entre refutação e falsificação pode ser resolvida com base nas propriedades da cifra assimétrica empregada para assinaturas. Nele, cada chave pública possui muitas inversas, sendo uma escolhida como chave privada.

  Em disputas, a assinatura da mensagem de autoria contestada deve ser reemitida em juízo. Se tal reemissão diferir da assinatura em disputa, a assinatura questionada teria quase certamente sido produzida com chave obtida por ataque ao sistema criptográfico via PLD (descrito em XIX.b).

- **Assinaturas em grupo:** (Chaum, COMPUSEC '91)

  Neste protocolo ajuizado, cada agente pertence a um grupo, a assinatura é individual mas a identidade do signatário fica ofuscada na verificação pública, que identifica apenas o grupo a que ele pertence. Em caso de disputa ou diante de eventual necessidade, a verificação ajuizada da assinatura é executada e a identidade do signatário é revelada.

PLD e Fatoração aplicados à Esteganografia

- **Esteganografia: Canal Subliminar** (Simmons, CRIPTO '83)

  Um *canal subliminar* é um mecanismo que permite a transmissão de uma mensagem secreta dentro de uma mensagem inócua. Ao invés de chave e/ou algoritmo, o próprio uso da criptografia é ocultado pelos principais.
  Esquemas de assinatura digital via de regra permitem a introdução de canais subliminares na assinatura lavrada em mensagens inócuas, onde a mensagem secreta é ocultada através da escolha da chave pública de verificação, ou na preparação do autenticador (Cap. 5).
  Simmons descreve variantes de algoritmos assimétricos para assinatura que seriam imunes ao estabelecimento de canais subliminares por essas vias.

- **Criptografia Homomórfica**

  (ou, processamento de dados encriptados):

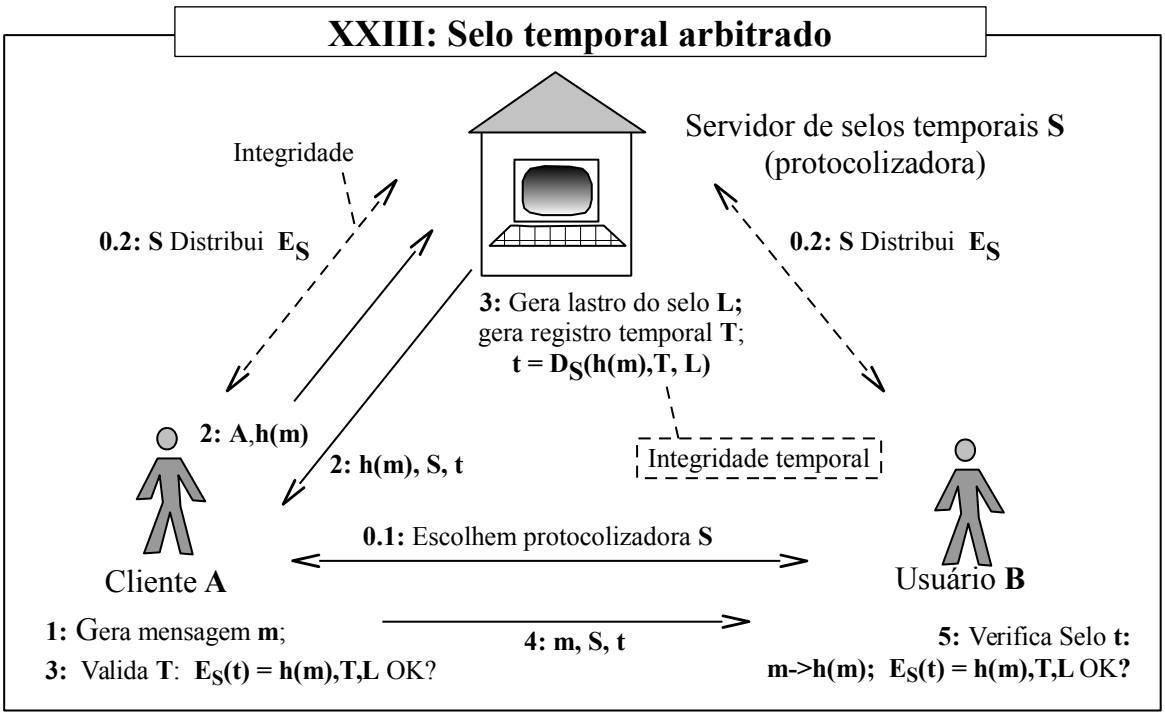
  Existem situações onde se pretende processar dados sensíveis sem revelar, na entrada do processo, esses dados (por exemplo, quando se aluga tempo em máquinas de terceiros). Isto é: representando o processo por **f( )**, deseja-se uma cifra **k( )** que comute com **f.** [que satisfaça **f(k(_) = k(f(_))]**

  (Feigenbaum, EURO '85)**:** Existem algumas funções para as quais é possível calcular **f(x)** ocultando **x**. O problema de se determinar quais funções são essas é conhecido como "*o problema do ocultamento de informação a um oráculo*".

  Para se acompanhar a evolução no estudo desse problema, a respectiva página da Wikipédia é um bom ponto de partida: https://en.wikipedia.org/wiki/Homomorphic_encryption

Serviços de validação temporal

- **Selo temporal assinado por TTP**



Premissas, propósito e limitações do protocolo XXIII:

a- A protocolizadora provê marcos (selos) temporais para a existência de documentos digitais, assinando uma relação entre um registro temporal e um índice (hash criptográfico) do documento.

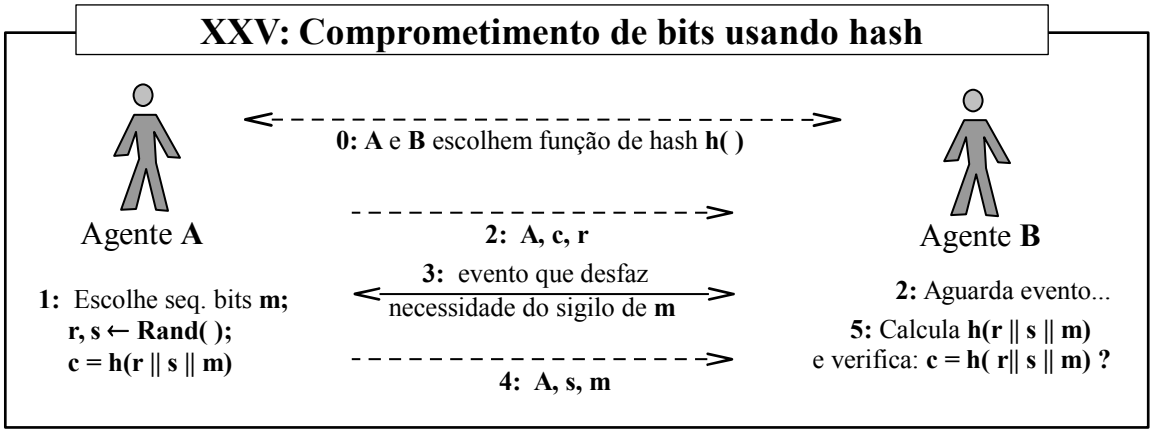
b- Tal tipo de protocolo presume integridade e corretude do registro temporal do momento da emissão do selo. Garantias desta presunção podem ser oferecidas por um lastro, geralmente formado por árvore ou lista de tamanho fixo, com os últimos selos emitidos pelo mesmo TTP.

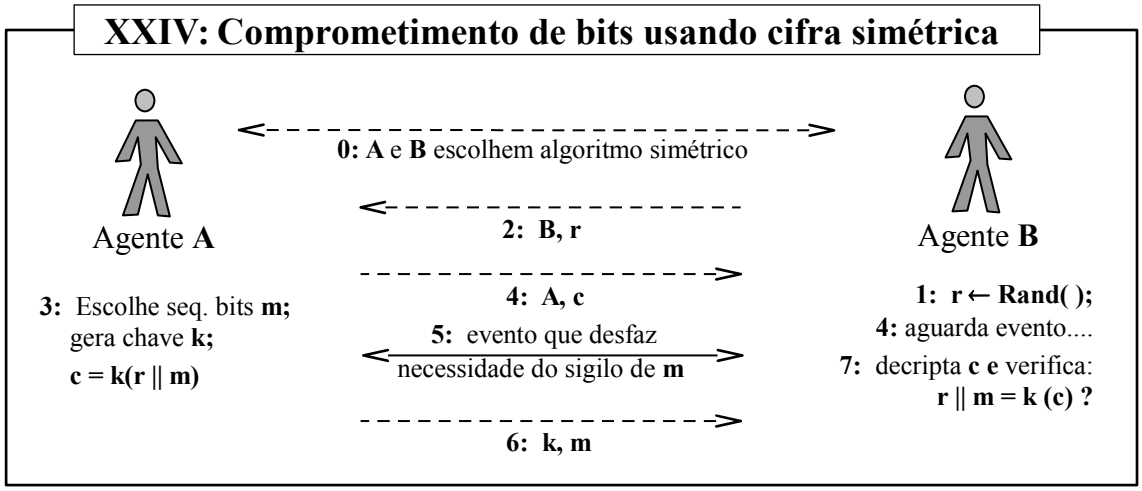
c - Ao emitir selo temporal para documentos o TTP não invade a pri-vacidade do principal, pois conhece apenas o resumo do documento.

d- Com o advento dos blockchains de acesso público e manutenção distribuída, como o iniciado em 2009 pela Bitcoin, e os de propósito geral do consórcio Ethereum, iniciado em 2013, os serviços de protocolização temporal arbitrado por TTP tornaram-se obsoletos.

Comprometimento de bits (aposta encoberta)

- **Protocolos para pré-comprovação de uma escolha sem revelá-la.**





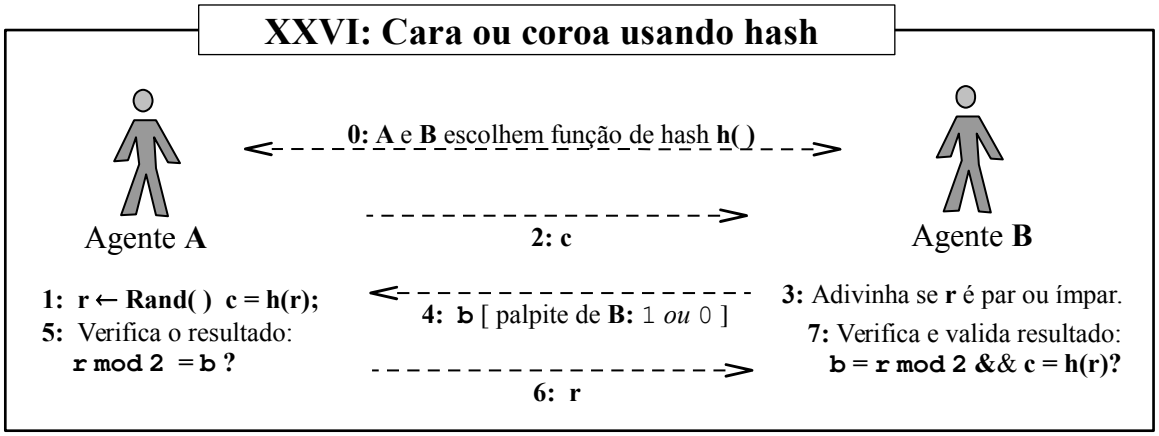
Propósito, premissas e limitações dos protocolos XXIV, XXV:

a- As comunicações antes e depois do evento (XXIV.3, XXV.5) pressupõem integridade de origem, se o propósito incluir detecção de trapaças. posteriormente uma dentre duas ou mais respostas diferentes a apresentar)

b- Com tal propósito, **r** na transmissão **2** serve para evitar que **A** tente burlar o comprometimento, controlando o custo de ataques de aniversário anterior ao passo 2 que lhe permita (a A) escolher qual **m** (dentre dois ou mais) revelar no passo 4. Por sua vez, **s** controla o custo de ataque de dicionário que permita a B deduzir **m** de **c** antes do passo 3.

Aplicações do Comprometimento de Bits

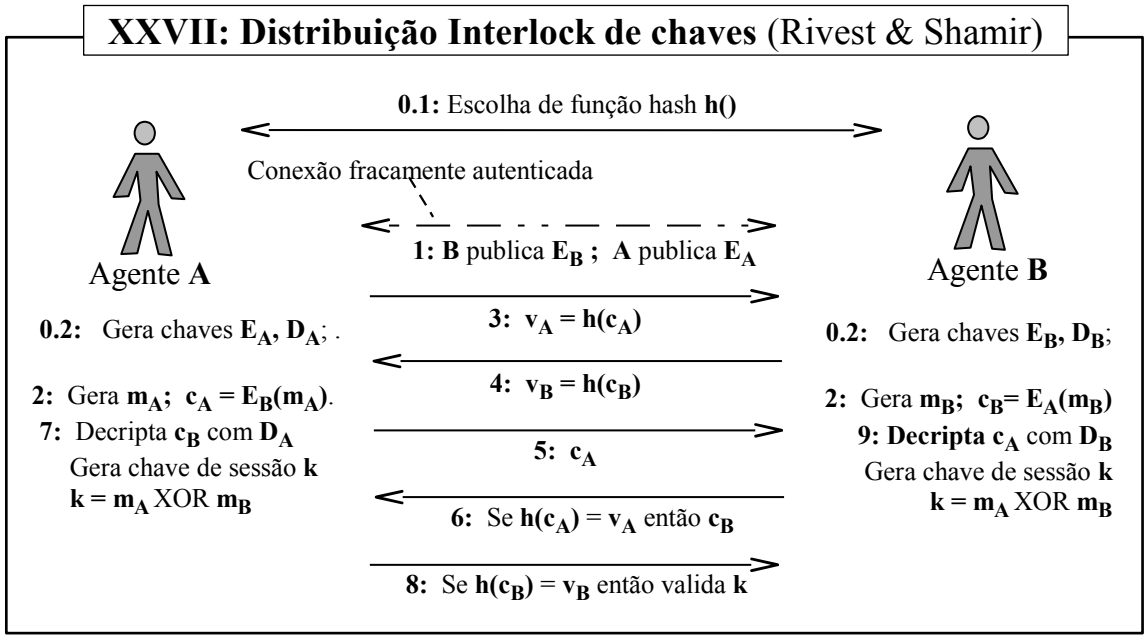
Cara ou Corôa: Protocolo combinatório para decidir entre opostos.



Propósito, premissas e limitações do protocolo XXVI:

- **a-** Trata-se de um caso particular de comprometimento de bits, no qual o evento que desfaz a necessidade de sigilo da escolha de **A** (cara ou coroa, par ou ímpar) é o palpite de **B** sobre esta escolha. O protocolo decide a favor de **B** se **B** acertar o palpite, e a favor de **A** se **B** errar.
- **b-** As premissas de confiança deste protocolo são as mesmas dos protocolos para comprometimento de bits (XXV.a-b).
- **c-** Este protocolo evidencia séria limitação do comprometimento de bits, decorrente da natureza do seu propósito: o de resolver uma corrida crítica. Quem primeiro conhecer o resultado leva vantagem, pois pode trapacear negando ao oponente tal conhecimento. E qualquer tentativa de neutralizá-la a torna recursiva, pois simultaneidade é conceito alheio.
- **d-** síntese: "simultaneidade" não cabe nem como premissa de confiança nesse tipo de protocolo.

Interlock: Detectando MITM ao distribuir chaves públicas-



Premissas do protocolo Interlock:

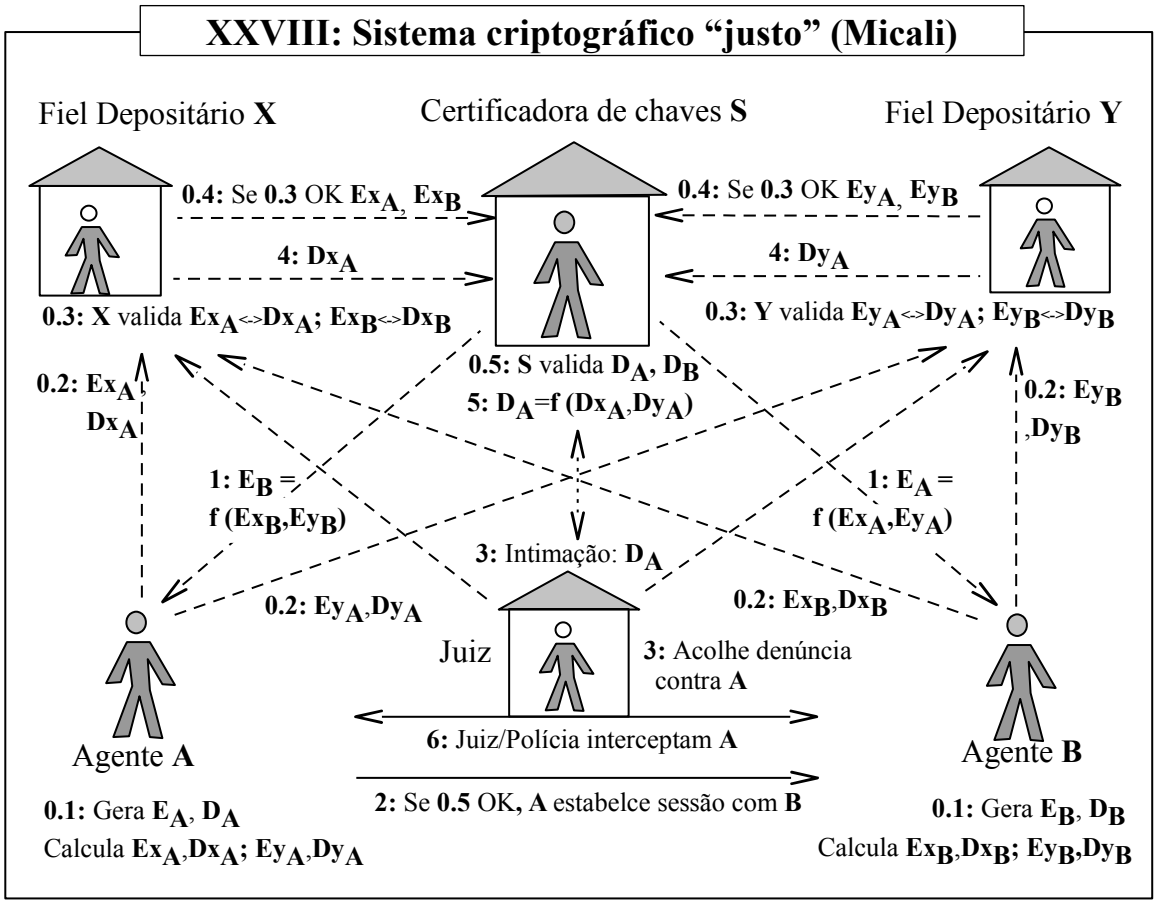
a - Um *man in the middle* **X** não terá como retransmitir ou alterar $\mathbf{m_A}$ (VI.b.5-7) sem ter sua ação detectada em **8**, pois não saberá como produzir **m'** tal que $\mathbf{h(E_X(m_A)) = h(E_B(m'))}$, embora possa substituir $\mathbf{v_A}$ e $\mathbf{c_A}$ nos passos **3** e **5**.

b - **X** não terá como retransmitir ou alterar $\mathbf{m_B}$ (VI.b.8-10) sem ser detectado em **8**, pois não saberá produzir **m'** tal que $\mathbf{h(E_X(m_B)) = h(E_A(m'))}$, embora possa substituir $\mathbf{v_B}$ e $\mathbf{c_B}$. **X** poderá portanto tentar "plantar um diálogo", mas não poderá fazer escuta ativa

c - Assim como as variantes que buscam superar as limitações do protocolo Diffie & Hellman em relação à premissa de prévia autenticação precípua dos principais, o Interlock tornou-se obsoleto com a popularização do uso do SSL e certificados digitais X.509.

Custódia de Chaves (*key escrow*)

- **Protocolos para caução compartilhada de chaves privadas**



Premissas, propósito e limitações do protocolo XXVIII:

a- O protocolo busca controlar a quebra de sigilo de chaves privadas sob demanda judicial. As chaves do par são divididas em partes, destinadas a distintos custodiantes, onde cada partição pode ser individualmente validada (ex: $\mathbf{Dx_A}$<->$\mathbf{Ex_A}$ **,** passo 0.**3**). Uma chave só pode ser reconstruída (por **f**) com todas as partes presentes. A escritura da chave pode ser validada, quanto a completeza das custódias, a partir dos verificadores (**0.4).** Implementado em firmware (projeto clipper), chaves privadas só seriam habilitadas se estiverem escrituradas (**0.5**). Após 2001, com o PATRIOT Act (EUA), protocolos para esse tipo de custódia se tornaram obsoletos.

b- Custódia compartilhada ou distribuída entre dispositivos ou agentes com interesses alinhados é implementada por protocolos do tipo "s*plit-key*".

Questões éticas sobre custódia de chaves

- ## Crítica de Bruce Schneier:

Além dos planos do governo dos EUA de estabelecer a custódia de chaves privadas como padrão, há várias propostas comerciais em oferta no mercado. Quais são as vantagens da custódia de chaves para o usuário? Bem, não há realmente nenhuma. O usuário não ganha nada da custódia que não poderia prover por si mesmo. A caução garante que a polícia pode perscrutar suas linhas de voz ou ler seus arquivos, apesar de criptografados. Garante que a agência NSA pode perscrutar ligações internacionais mesmo sem mandato judicial, embora encriptadas. Talvez a custódia possa permitir o uso da criptografia em países que hoje a proíbem, mas esta parece ser a única vantagem. A custódia de chaves criptográficas tem inúmeras desvantagens. O usuário tem que confiar na segurança dos procedimentos das agências cartoriais, depositários fiéis de sua chave privada em caução, e na lisura das pessoas envolvidas. Tem que confiar que os agentes da caução não irão modificar suas regras de conduta, que o governo não mudará as leis sobre custódia, e que as pessoas com poder para recuperar sua chave privada o farão apenas dentro da lei. (proposta de lei McCain-Kerrey no senado dos EUA)

É difícil imaginar este cenário funcionando idealmente, sem nenhum tipo de pressão legal ou política. O próximo passo óbvio seria a proibição da criptografia assimétrica sem custódia de chaves privadas. Essa seria possivelmente a única forma de garantir o retorno no investimento em um desses sistemas, e de obrigar organizações criminosas sofisticadas a usarem sistemas próprios. Não está claro quão difícil será fazer cumprir uma tal lei. Os pesquisadores que trabalham em criptografia terão que ter licença especial para manipular sistemas não custodiados? Como serão testadas implementações de novos sistemas em software, já que os sistemas custodiados envolvem a participação do hardware no protocolo?

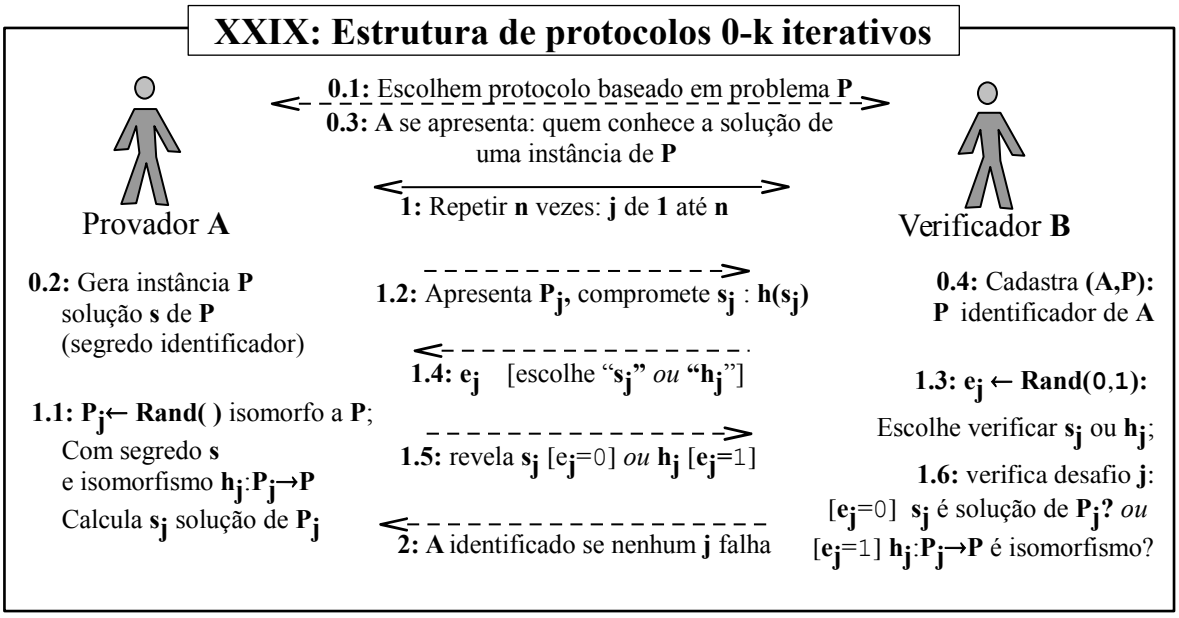
E há também questões legais. Como a escritura caução a imputabilidade do usuário, em caso de vazamento? Haverá uma hipótese implícita de que se algum segredo vazar pela conta do usuário, então certamente terá sido pelo usuário e nunca através dos agentes cartoriais? E se o banco de dados do cartório for roubado? E se este roubo for acobertado pelo governo? Ainda mais perigoso é o uso das cauções em espionagem para fins políticos.

Se as chaves de assinatura forem também custodiadas, existem problemas adicionais. Seriam aceitos em juízo documentos assinados com chave custodiada como prova contra o proprietário da chave? A globalização da criptografia traria problemas adicionais. E se governos de outros países não aceitarem como de fé pública os agentes cartoriais americanos? As companias multinacionais teriam que aderir a sistemas e normas diferentes em cada país onde operam? Se um país obrigar o usuário a escriturar sua chave de assinatura, poderá ele refutar em outro país sua assinatura num contrato, alegando ter sido forjada por agentes cartoriais onde foi obrigado a escriturar sua chave? E os países que patrocinam espionagem industrial em benefício de suas empresas? Certamente não irão escriturar suas chaves em outros países. As comunicações digitais oferecem oportunidade bem mais ampla para atividades de monitoramento das ações, opiniões, compras, associações, etc., de cidadãos do que seria possível no mundo analógico. Tornar esta capacidade quase automática e não rastreável pode oferecer uma tentação a mais para estados com tendências não democráticas.

Provas com conhecimento zero: 0-k(*nowledge*)

- **Protocolos 0-k iterativos -** um provador tenta convencer um verificador de que detém um dado secreto que o identifica, sem necessidade de compartilhar esse dado (geralmente um dado que descreve a solução de uma instância de um problema matemático). São protocolos baseados no método de Monte Carlo (ver Apêndice A-18), que estipula uma série de desafios em rodadas onde, enquanto acertados os desafios, acumulam probabilidades para tal convencimento, ou o anulam caso haja erro na verificação de alguma rodada.

XXIX: Estrutura de protocolos 0-k iterativos



Premissas e limitações dos protocolos 0-k iterativos:

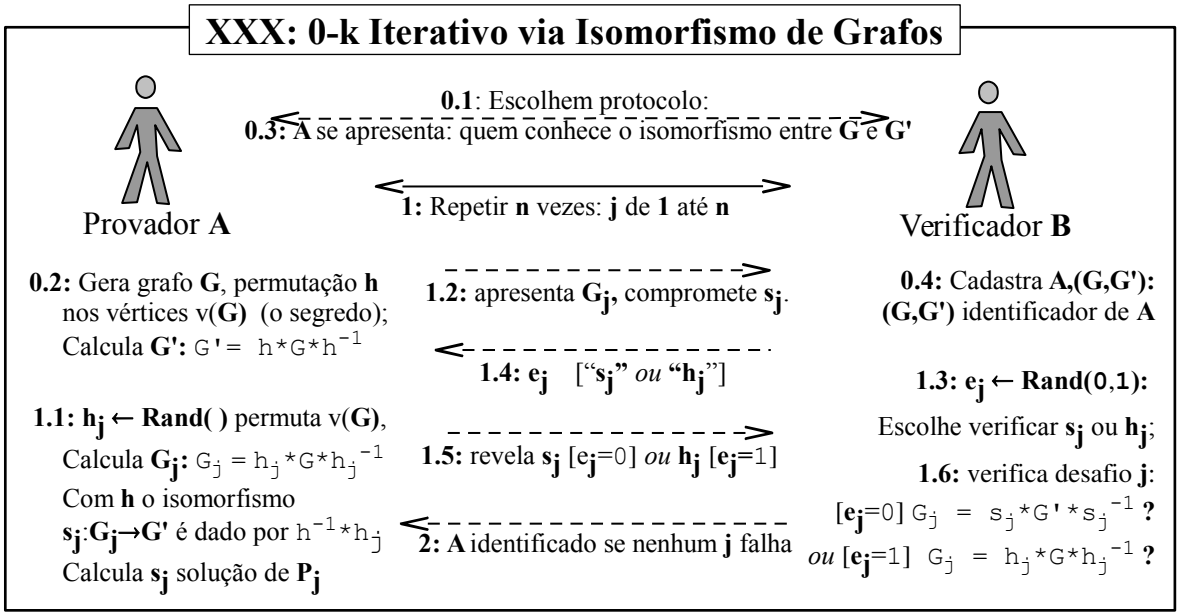
a - Só o provador **A** deve conhecer **s.** Se **A** não conhece **s**, poderá apresentar apenas uma "parte metade" correta (ou **hj** ou **sj**) de cada desafio. Aí há probabilidade = 1/2 de falha em **1.6**, a qual indicaria fraude do provador; após **n** rodadas respondidas corretamente, = $1/2^n$ (Apêndice A-16)

b - A escolha aleatória em **1.3** em tempo real pelo verificador é essencial, pois esse tipo de protocolo é vulnerável a espelhamento e conluio (serve para autenticação subjetiva). Como pretende identificar o provador requerendo integridade do canal entre ambos, serve em sistemas fechados.

Exemplo de protocolo 0-k

- **Baseado no problema de Isomorfismo entre Grafos -**

O problema em que se baseia esse protocolo é o de, dados dois grafos, decidir se são isomorfos ou não, e qual o isomorfismo (ver Apêndice A-16,17).



Detalhes do protocolo XXX:

a - A complexidade da instância do problema de isomorfismo em que se baseia a escolha dos dados de identificação do provador inviabiliza a fraude pelo provador àquelas descritas em XXIX.a se os parâmetros da instância forem adequadas (v(G)~300, densidade arestas ≅ v(G)/2, etc)

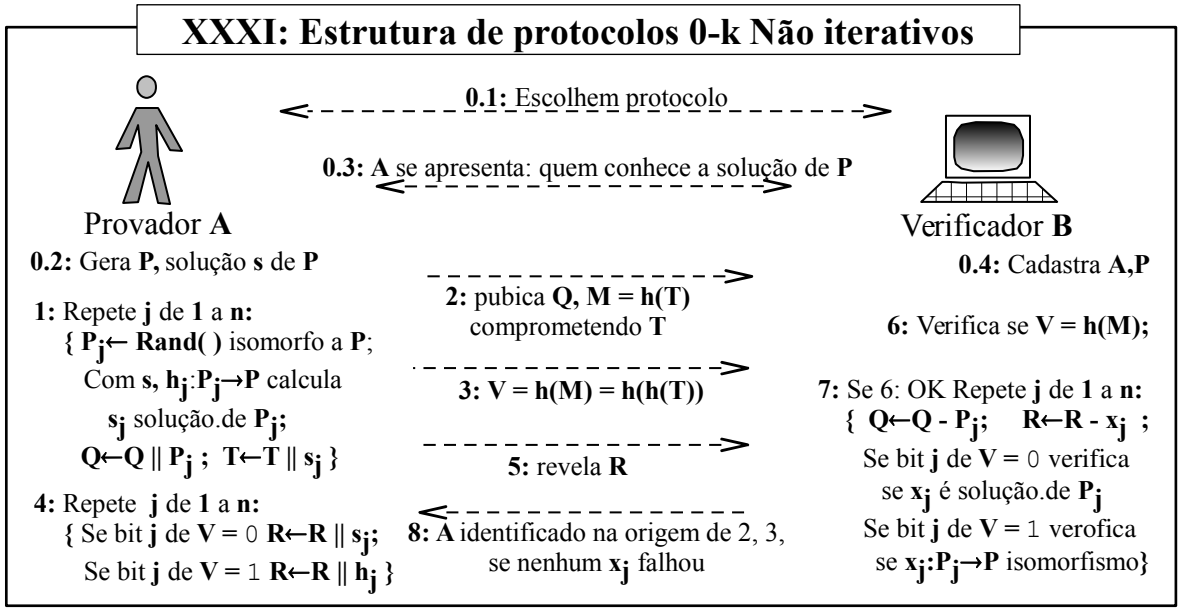
b - As operações matemáticas necessárias para as rodadas do protocolo neste caso se restringem à multiplicação de matrizes e geração de permutações randômicas (ver Apêndice A-16,17)

c- Qualquer teorema matemático pode representar-se por um grafo, onde a demonstração é codificada como um ciclo hamiltoniano, e a busca desse ciclo substituir a do isomorfismo no protocolo (Blum, 1986).

Protocolos 0-k Não iterativos

Num protocolo 0-k iterativo, o verificador se convence da identidade do provador por ser o agente ativo na escolha aleatória da "parte metade" de cada desafio a verificar (**1.3**). Esta ação randomizadora pode ser substituída por uma forma de comprometimento de bits, que inviabiliza ataques de espelhamento ou conluio envolvendo o provador, em implementações para autenticação *off-line.*

XXXI: Estrutura de protocolos 0-k Não iterativos



Premissas e limitações dos protocolos 0-k não iterativos:

a - Se **A** não detém **s**, terá dificuldade para pre-calcular uma sequência **V** produzível de **T** pela função de hash **h** que o possibilite fraudar cada desafio **j** sabendo de antemão qual "metade" do desafio terá que mostrar em **R**, pois cada variante $\mathbf{P_j}$ e isomorfismo $\mathbf{s_j}$ estão comprometidos em **V**, de onde os primeiros bits simulam a escolha aleatória do provador (indicando qual "metade" da "rodada" este quer verificar).

b - O número **n** de repetições (equivalente a rodadas) que torna a versão não interativa equiparável à interativa quanto a probabilidade de sucesso de fraude do provador, deve ser maior que na versão iterativa, para se evitar a chance de pré-cálculo da sequência de escolhas.

c- As premissas de confiança são as mesmas que as de versões interativas menos a necessidade de o verificador randomizar escolhas.

Protocolos esotéricos

- **Transmissão ou assinatura cega de mensagens -** (Chaum)

  Um agente envia várias mensagens encriptadas ao receptor, que escolhe apenas uma para decriptar, e o emissor não saberá qual. No caso da assinatura, o signatário não decriptará apenas uma, a que será assinada.

- **Assinatura de contratos -** (Even, Goldreich & Lempel)

  Existem protocolos iterativos para simular assinaturas "simultâneas" (onde a ordem em que foram lavradas é desconhecível) dos signatários de um instrumento contratual.

- **Eleições eletrônicas -** (Chaum, 2003, 2009, 2016)

  O sigilo do voto (que torna a autoria do voto desconhecível na apuração) é um propósito dificilmente não conflitante com o propósito da integridade da apuração. Existem protocolos para eleições que autenticam probabilisticamente os votantes e a soma de votos (por aplicações do método de Monte Carlo), com propriedades de proteção à anonimidade do eleitor, à verificação pelo eleitor da correta tabulação de seu voto, e contra a duplicação, substituição ou adulteração de votos.

- **Moeda eletrônica -** (Chaum, Fiat & Naor; e outros)

  Cartões de crédito, como cheque, permitem rastreamento de transações. Existem protocolos complexos que permitem transações anônimas (autenticáveis mas não rastreáveis à origem), com controle de modalidade de transferência. O uso destes protocolos depende da disposição de agentes financeiros e reguladores em aceitá-las em permuta por moeda fiat. Limita-os, o controle de cópia. As criptomoedas, por sua vez, lastreadas em cadeias de blocos de transações autenticadas em vários níveis, inaugura a possibilidade de controle descentralizável para emissão e contabilidade de moeda ou dinheiro.

4: Técnicas Criptográficas

- **Custo de plataforma x Tempo médio estimados**

  Para quebra por força bruta de chaves de algoritmos simétricos robustos – (supondo 10^6 cifragens/segundo por CPU, ao custo de hardware em 1995)

1995-Custo hardware	Comprimento útil da chave secreta					
	40 bits	**56 bits**	**64 bits**	**80 bits**	**112 bits**	**128 bits**
US$100 mil	2 segundos	35 horas	1 ano	$7x10^4$ anos	10^{14} anos	10^{19} anos
1 milhão	0.2 seg	3.5 horas	37 dias	$7x10^3$ anos	10^{13} anos	10^{18} anos
10 milhões	0.02 seg	21 min	4 dias	700 anos	10^{12} anos	10^{17} anos
100 milhões	2 miliseg	2 minutos	9 horas	70 anos	10^{11} anos	10^{16} anos
1 bilhão	0.2 miliseg	13 seg	1 hora	7 anos	10^{10} anos	10^{15} anos
10 bilhões	0.02 mseg	1 segundo	5.4 min	245 dias	10^9 anos	10^{14} anos
100 bilhões	2 microseg	0.1 seg	32 seg	24 dias	10^8 anos	10^{13} anos
1 trilhão	0.2 μ seg	0.01 seg	3 seg	2.4 dias	10^7 anos	10^{12} anos
10 trilhões	0.02 μ seg	1 miliseg	0.3 seg	6 horas	10^6 anos	10^{11} anos

Robusto: termo técnico que significa não ser conhecido nenhum método de ataque sobre critpograma C com $|C| > U_K$ (pag. 19) mais eficiente do que a força bruta, isto é, do que a busca exaustiva num espaço de chaves equiprováveis

- **Estimativas para tamanho de chaves RSA sadias -**

  Extrapolando dados históricos sobre a capacidade de fatoração de inteiros, considerando projeções segundo a "lei de Moore"

  (Rivest: estimativa otimista em 1990 – Schneier: estimativa em 1995)

Ano	Comprimento de chaves públicas RSA em bits, para proteção contra		
	Indivíduos	**Grandes corporações**	**Grandes estados**
1995	405 - 768	542 - 1280	1399 - 1536
2000	422 - 1024	572 - 1280	1512 - 1536
2005	439 - 1280	602 - 1536	1628 - 2048
2010	455 - 1280	631 - 1536	1754 - 2048
2015	472 - 1536	661 - 2048	1884 - 2048

Critérios para escolha de chaves

- **Relação aproximada entre tamanhos de chaves**

Mesmo nível de custo de quebra estimado - (Schneier, 1995)

Tipo de algoritmo	**Ataque p/ força bruta (S) e fatoração de inteiros (A)**				
Simétrico (R. Feistel)	56 bits	64 bits	80 bits	112 bits	128 bits
Assimétrico RSA	384 bits	512 bits	768 bits	1792 bits	2304 bits

- **Tamanho |K| dos espaços de senhas (chaves mneumônicas): -**

Subconjunto de caracteres	**Comprimento da senha**				
	4 bytes	5 bytes	6 bytes	7 bytes	8 bytes
letras minúsculas (26)	460 000	$1.2*10^{7}$	$3.1*10^{8}$	$8.0*10^{9}$	$2.1*10^{11}$
minúsculas+dígitos (36)	$1.7*10^{6}$	$6.0*10^{7}$	$2.2*10^{9}$	$7.8*10^{10}$	$2.8*10^{12}$
alfanuméricos (62)	$1.5*10^{7}$	$9.2*10^{8}$	$5.7*10^{10}$	$3.5*10^{12}$	$2.2*10^{14}$
imprimíveis (95)	$8.1*10^{7}$	$7.7*10^{9}$	$7.4*10^{11}$	$7.0*10^{13}$	$6.6*10^{15}$
caracteres ASCII (128)	$2.7*10^{8}$	$3.4*10^{10}$	$4.4*10^{12}$	$5.6*10^{14}$	$7.2*10^{16}$
ASCII estendido (256)	$4.3*10^{9}$	$1.1*10^{12}$	$2.8*10^{14}$	$7.2*10^{16}$	$1.8*10^{19}$

- **Tempo médio de busca exaustiva em espaços de senhas -**

10^{6} verificações/seg	**Comprimento da senha**				
Subconjunto ASCII	4 bytes	5 bytes	6 bytes	7 bytes	8 bytes
letras minúsculas (26)	0.5 seg	12 seg	5 min	2.2 horas	2.4 dias
minúsculas+dígitos (36)	1.7 seg	1 min	36 min	22 horas	33 dias
alfanuméricos (62)	15 seg	15 min	16 horas	41 dias	6.9 anos
imprimíveis (95)	1.4 min	2.1 horas	8.5 dias	2.2 anos	210 anos
caracteres ASCII (128)	5.5 min	9.5 horas	51 dias	18 anos	2300 anos
ASCII estendido (256)	1.2 horas	13 dias	8.9 anos	2300 anos	$6 \cdot 10^{5}$ anos

Ataques de Dicionário geralmente combinam busca exaustiva de possíveis senhas começando pelas menores, com busca entre palavras do léxico ou combinações com sonoridade semelhante.

Primitivas de algoritmos assimétricos

- **Implementação eficiente da função de exponenciação modular**

$$\text{expmod}(a,x,m) = a^x \bmod m$$

```
/*                    função expmod                    */
/* uso: base a expoente x modulo m de tipo array int */
/* supõe m>0, x≥0, aritmética sobrecarregada p/extint*/
    exint r, s, y;
    exint expmod (exint *a, exint *x, exint *m)
    {
        r = 1; y = x; s = a%m;
            while( y ) {
            if( y&1 )
                r = (r*s)%m;
            s = (s*s)%m;
            y = y>>1
        }
        return( r )
    }
```

- **Inversão da exponenciação modular: PLD (logaritmo discreto)**

  O algoritmo mais eficiente hoje conhecido para resolver o PLD são o algoritmo de Pohlig, Hellman & Silver (PHS) ou um algoritmo para cálculo de índices (ver Koblitz, N. "*A Course in Number Theory and Cryptography*"), ambos de custo exponencial no tamanho do maior fator do índice de Euler do módulo (ver definição de "função de Euler" em Koblitz ou na wikipedia)

- ## Inversão do produto de inteiros: fatoração

Os algoritmos mais eficientes hoje conhecidos para fatoração são o Crivo Gaussiano (QS) e o Crivo de Corpo Numérico (NFS), de complexidades:

QS: Tempo de execução $T \approx e^{(1+c*|p|*\ln(|p|))^{1/2}}$ ($|p| <\sim 360$);

NFS: $T \approx e^{(1.923+c*|p|*\ln^2(|p|))^{1/3}}$ (mais eficiente p/ $|p| >\sim 360$)

Onde $|p|$ é o número de bits do número p a ser fatorado.

O problema da fatoração de inteiros é estudado desde Euclides (450A.C.). Tais algoritmos transportam o problema para um corpo de extensão de Galois $GF(q^r)$, com mais estrutura que Z_n, onde possíveis fatores são representados por polinômios. Exemplo:

$p = 2^{113}-1 = 3391*23279*65993*1868569*1066818132868207;\ |p|=112$

- ## Histórico dos limites práticos da fatoração -

Inteiros *duros* ($n = p*q$ onde p, q são primos *duros* e $|p| \approx |q|$):

Ano	*Tamanho de* **n**	*Exemplo / Complexidade e técnica empregada*
	primo p é *duro* se $p-1$ tem um fator "grande" (de tamanho $\approx |p|$)	
1970	41 dígitos	Vários meses de um grande porte, usando QS.
1991	Série RSA	Desafio lançado pela RSA: $\mathbf{n_i}$ de 100 a 500 dígitos; $1 \le i \le 42$
1993	RSA-120 ($\mathbf{n_3}$)	825 mips_ano, em 3 meses usando QS.
1994	RSA-129 ($\mathbf{n_4}$) (428 bits)	~5000 mips_ano, em 8 meses usando QS distribuído *(em 1600 máquinas de voluntários na Internet, via e-mail).
1997	154 dígitos (512 bits)	Supõe-se fatorável pelo Crivo de Corpo Numérico (NFS), em poucos meses com alguns milhões de US$.

Números de Fermat e de Mersenne

(moles ou grandes demais para utilidade criptográfica):

ordem	**Número**	**Fermat: $2^{2^n}+1$ Mersenne: 2^p-1;**
9º F.	$2^{512}+1$	Composto: decomposição encontrada usando NFS.
41º M.	$2^{24036583}-1$	Primo: maior dos conhecidos (em 15 de maiio de 2004), com mais de 7.3 milhões de dígitos

* (mip_ano = 1 milhão de instruções por segundo durante 1 ano)

- **Algoritmo probabilístico para teste de primalidade: Lehmann -**

```
/* uso: função rand( ) geradora de números randomicos*/
/* m parâmetro de incerteza sobre a primalidade de p */
/* p candidato a primo. aritmética estendida (extint)*/
/*   lehmann() retorna probabilidade de p ser primo  */
int lehmann (exint *p, unsigned int j)
    {   exint a, r; int res;  res=1;
        while(j && res) {
            j-= 1; a= rand(p-1);      /*a < p*/
            r= expmod(a,(p-1)/2,p);
            if(!(r==1) && !(r==p-1)) res= 0;
        }   return(res)
    /*r testa se p viola equação de Fermat*/
    }
```

- **Heurística para geração de primos grandes** (ver Apêndice A-18)

  Algoritmos tipo Monte Carlo para testar primalidade acumulam probabilidades de que `p` é primo, ou deduzem que `p` é composto por violar o teorema de Fermat ($a < p \Rightarrow a^{p-1} \bmod p = 1$). No algoritmo de Lehmann acima:

  - Se algum teste concluir que `p` é composto, `lehmann(p,j)` retorna `0`
  - Após `j` testes sem falha, retorna `1`, com probabilidade de `p` ser primo $\cong 1 - (\ln(2)*|p|-2)/(\ln(2)*|p|-2+2^{j+1})$ (pelo t. de Bayes)

  onde $|p|$ = número de bits de `p`, e os testes com resíduos `a` são feitos pela raiz quadrada da equação de Fermat. O algoritmo de Lehmann se baseia em:

  1) - Proporção dos resíduos `a` que violam o teorema de Fermat em $\mathbb{Z}_p \geq 50\%$ se `p` for composto (os testes são feitos com símbolos de Jacobi devido aos números de Carmichael, conforme explicado no Apêndice C-3.4)

  2)- Distribuição dos primos em $\mathbb{N}$: Pr(`p` é primo) $\cong 2/(\ln(2)*|p|)$

Geração de primos para criptografia assimétrica

- **Algoritmo probabilístico para geração de primos com `t` bits -**

```
/*     função geraprimo() retorna primo de t bits     */
/*     uso: função rand() gera números randomicos     */
/* j parâmetro de precisão sobre a primalidade de p   */
/* t =|p| número de bits de p; aritmética estendida   */
    exint geraprimo (unsigned int t, unsigned int j)
    {
        exint p;
        p = (rand(t-1)<<1)+1;  /*p impar com t bits*/
        while(!lehmann(&p,j))
            p+=2;
        return(p)
    }
```

Um inteiro ímpar de `t` bits com alta probabilidade de ser primo é obtido a partir de um candidato aleatório de `t` bits: este e seus sucessores ímpares são submetidos a uma série aleatória de `j` testes com a raiz quadrada da equação do teorema de Fermat, até que um passe na série sem falhar (método de Monte-Carlo: A-18).

- **Exemplo de probabilidades com geraprimo( ) -**

Geração de `p` com Lehmann		**Prob. de `p` NÃO ser primo tendo passado `j` testes**			
número de bits de `p`	**número médio de candidatos testados**	**série com `j=10`**	**série com `j=20`**	**série com `j=30`**	**série com `j=40`**
t=128	44	0.042	0.41×10^{-6}	0.40×10^{-9}	0.39×10^{-12}
t=256	89	0.079	0.84×10^{-6}	0.81×10^{-9}	0.79×10^{-12}
t=384	133	0.114	0.12×10^{-5}	0.12×10^{-8}	0.12×10^{-11}
t=512	177	0.146	0.16×10^{-5}	0.16×10^{-8}	0.16×10^{-11}
t=768	266	0.209	0.23×10^{-5}	0.24×10^{-8}	0.24×10^{-11}
t=1024	336	0.246	0.32×10^{-5}	0.31×10^{-8}	0.30×10^{-11}

Mecanismos e modos para construção de cifras

• Classificação de cifras quanto a parâmetros de entrada/saida-

1 - **Cifras de bloco** (***block ciphers***)**:**

- *Mensagem e criptograma são concatenações de unidades* (blocos) *de e/s, de tamalho fixo equivalente ao tamanho da chave.* (ex: **64** bits)
- *O algoritmo da cifra não armazena informação sobre o estado da cifragem de blocos anteriores. A execução pode ser feita em paralelo nos blocos da mensagem ou do criptograma.*
- *Cifra simula monoalfabética com alfabeto intratável* (ex: $\mathbf{2^{64}}$ blocos)

2 - **Cifras encadeadas** (***stream ciphers***)**:**

- *Mensagem e criptograma são concatenações de unidades de e/s, de tamalho fixo equivalente a fração do tamanho da chave.* (bit, byte)
- *O algoritmo da cifra precisa armazenar informação sobre o estado da cifragem do bloco anterior. A execução precisa ser sequencial.*
- *Cifra simula One Time Pad, exceto pela duração/tamanho da chave.*

• Modos de operação para cifras de bloco

1 - **Cifra direta:** *ECB Eletronic codebook* (opera como monoalfabética)

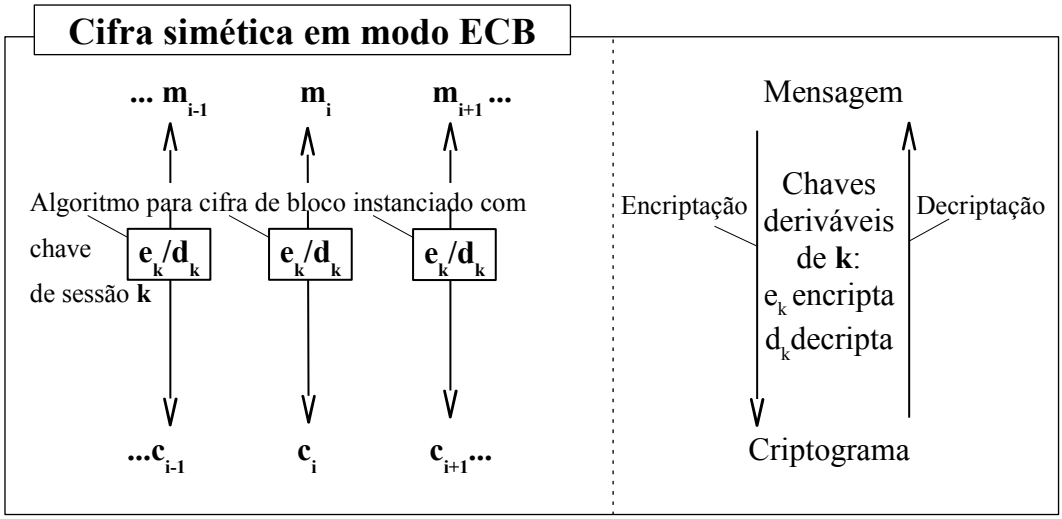
2 - **Cifra retroalimentada:** (opera em modo encadeado)

Modos de encadeamento (externos ao algortimo da cifra):

- *CBC ....Cipher block chaining*
- *CFB ....Cipher feedback chaining*
- *OFB.....Output feedback chaining*
- Outros ... *Galois Couter mode, etc.*

Modo ECB

- **$m=m_1||...||m_n$, cada bloco m_i é cifrado independentemente dos outros:**



Propriedades das cifras de bloco em modo ECB

a- Algorimos simétricos para cifra de bloco operam de modo mais simples possível em ECB. Paralelizável, é útil em aplicações onde a decriptação parcial não seqüencial é necessária, tais como para ban-cos de dados ou sistemas de arquivo em dispositivos de armazenagem

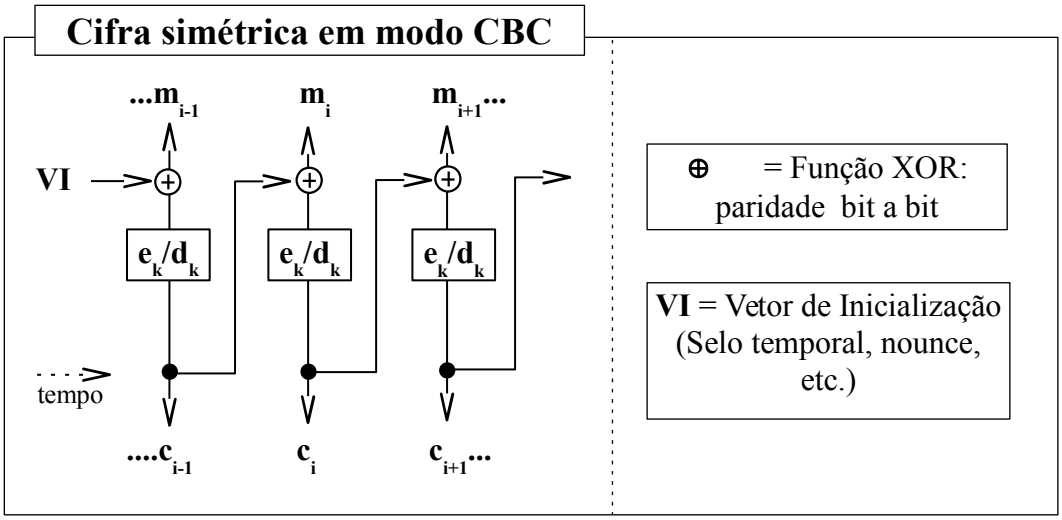
b - Erros de transmissão do criptograma invalidam o bloco onde ocorrem, mas não se propagam. Requerem enchimento no último blo-co, para tornar o comprimento de **m** múltiplo do tamanho do bloco.

c - Este modo é vulnerável a ataques por *replay* com substituição de blocos, onde o significado de um bloco do criptograma pode ser infe-rido sem o conhecimento da chave. Em geral, é o modo mais frágil.

d- Algoritmos assimétricos operam por default em ECB, pois esquemas onde são úteis -- envelope e assinatura -- têm sessões de um só bloco.

Modo CBC

- **Ao encriptar, o último bloco do criptograma é pré-misturado, via função XOR, ao próximo bloco da mensagem. Ao decriptar, é pós-misturado:**



Propriedades das cifras simétricas de bloco no modo CBC

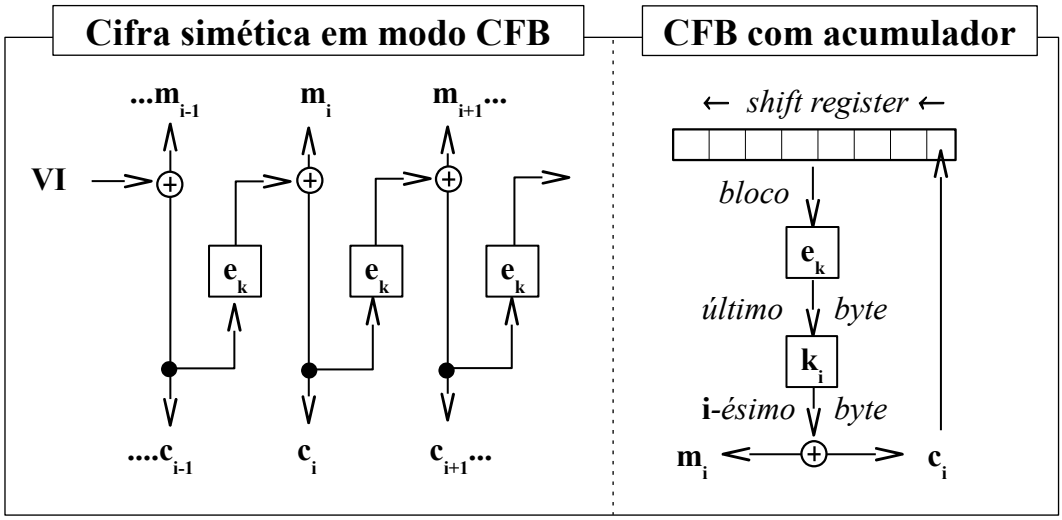
a - Modo mais adequado que ECB para transmissão em redes, útil em aplicações onde várias mensagens com trechos em comum ou de formato público são transmitidas em uma mesma sessão. O vetor VI pode ser enviado sem sigilo e deve ser único para cada mensagem.

b - Erros de transmissão do criptograma invalidam o bloco onde ocorrem, e o bit do bloco seguinte na posição onde ocorreu o erro. Não requer enchimento no último bloco da mensagem, podendo ser usado quando criptograma e mensagem devem ter mesmo tamanho.

c - Este modo evita ataques por escuta ativa com substituição de blocos, onde o significado de um bloco encriptado pode ser inferido por texto pleno anteriormente escolhido.

Modo CFB

- **O último bloco do criptograma é reencriptado, a saída é misturada ao próximo bloco da mensagem para cifrar, ou do criptograma p/ decifrar**



Propriedades das cifras simétricas de bloco em modo CFB

a - Algortimo sempre em modo encriptação, simulando gerador de OTP.

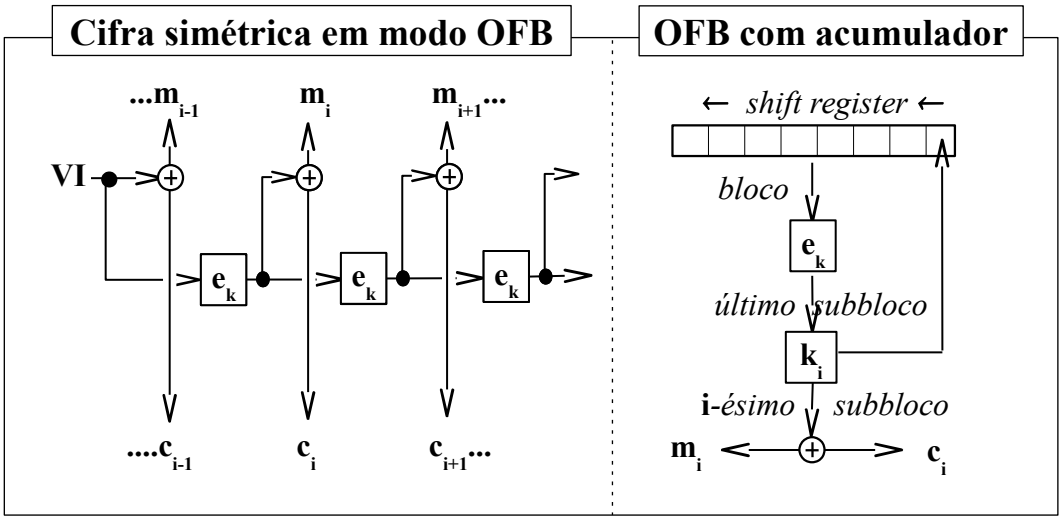
b- Modo útil para tráfego de rede e/ou em aplicações onde o mecanismo de transmissão usa palavras menores que o bloco da cifra. O vetor VI *precisa ser* único para cada chave e mensagem, sem requerer sigilo.

c - Erros de transmissão do criptograma invalidam o bit onde ocorrer o erro, e também o bloco (do acumulador) que o segue. Não requer enchimento no último bloco da mensagem, podendo ser usado quando criptograma e mensagem precisam ser do mesmo tamanho

d - Este modo não evita, mas detecta ataques por escuta ativa via substituição de bits, caso o texto pleno do bloco a ser manipulado seja conhecido do atacante. Valores de bits podem ser manipulados, à custa da recepção do bloco (do acumulador) seguinte ao bit.

Modo OFB

- **O último bloco de saída da reencriptação do VI é reencriptado, e a saída combinada ao próximo bloco da mensagem ou criptograma:**



Propriedades das cifras simétricas de bloco no modo OFB

a - Algortimo sempre em modo encriptação, simulando gerador de OTP.

b - Modo útil para encriptação *off-line*, em aplicações onde a transmissão é bem mais rápida que a execução do algoritmo de cifra. O vetor **VI** pode ser enviado sem sigilo, deve ser único para cada mensagem e precisa ser carregado na inicialização do acumulador.

c - Erros de transmissão do criptograma invalidam apenas o bit onde ocorrer o erro, tornando este modo indicado em aplicações de tempo real onde a tolerância a erros é mínima (voz, imagem, etc).

d - O subbbloco não deve ter tamanho (**b**) pequeno, menor que o do bloco da cifra, pois este modo simula gerador randômico para *one-time pad* iterando encriptações do **VI** onde o período médio é $\sim 2^b$.

Outros

- **Modo "Contador de Galois":**

  Consultar https://en.wikipedia.org/wiki/Galois/Counter_Mode

Construção de geradores de seqüências pseudo-randômicas e cifras encadeadas

- **Histórico -**

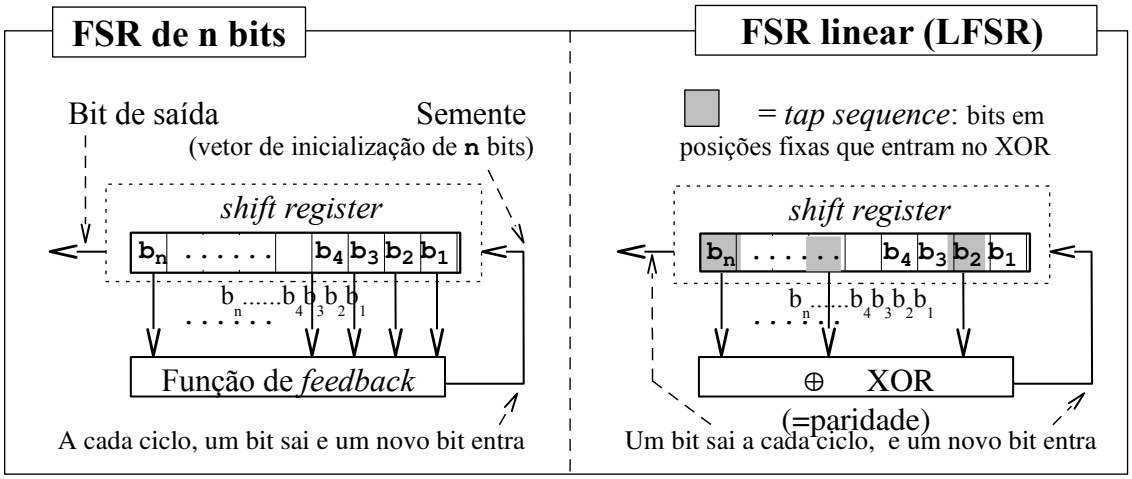
A teoria de projeto e análise de cifras encadeadas adaptou à eletrônica técnicas empregadas nas cifras de implementação mecânica até a segunda guerra mundial, através do uso de registradores de deslocamento *(shift registers)*. Essas técnicas formam, ainda, base importante da criptografia militar.

- **Registros de deslocamento com retroalimentação -**

Feedback Shift Registers (FSR) são os elementos básicos na maioria dos geradores de seqüências pseudo-randômicas utilizados na criptografia atual. São de implementação eletrônica simples, como também em software.

O *período* de uma seqüência gerada por um FSR de **n** bits, o n. de configurações subsequentes e distintas a partir da semente, é no máximo 2^n.

Os FSRs mais simples são os lineares (LFSR), que usam XOR como função de retroalimentação. As posições do registrador que participam no XOR formam a **seqüência de captura** (*tap sequence*) de um LFSR.



LFSRs de período máximo

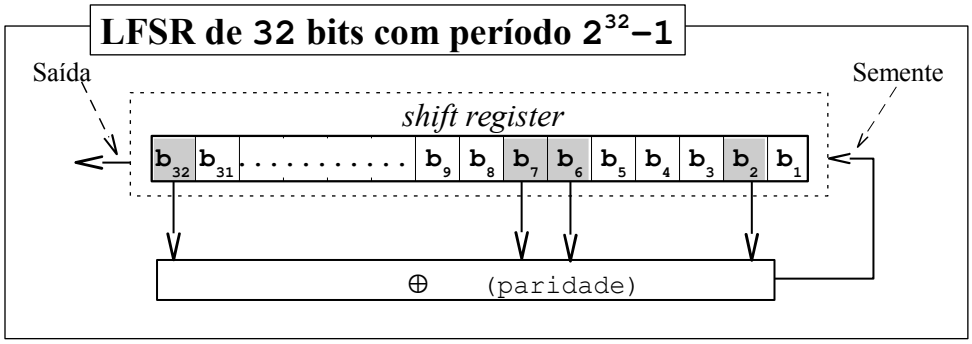
- **Escolha da seqüência de captura** (Selmer, 1965) **-**

  Para um LFSR ter período máximo (2^n-1), sua configuração com $\mathbf{b_i=1}$ nas posições de captura e $b_i=0$ nas demais deve ser tal que o polinômio $p(x) = \Sigma\, b_i x^i+1$, $1\le i\le n$ sobre Z_2 seja primitivo. Isto é, irredutível e satisfazendo

  - $[p(x)\,|\,(x^{(2^n-1)}+1)]$ && "d $[d|(2^n-1) \Rightarrow \emptyset(p(x)|(x^d+1))]$

- **Exemplo de construção de LFSR de período máximo -**

  Sabendo que $p(x) = x^{32}+x^7+x^6+x^2+1$ é primitivo sobre Z_2, um LFRS de **32** bits com captura nos bits 32, 7, 6 e 2 terá, portanto, período máximo.



Percorrerá a órbita máxima a partir de qualquer semente não nula. Com reenumeração dos bits (a partir de 0 em C), o LFSR será implementável em loop com

```
static unsigned long SR=1 /*ou qualquer semente≠ 0*/
int LFSR ( )    /*SR representa o shift register */
{
    SR=((((SR>>31)^(SR>>6)^(SR>>5)^(SR>>1))
        & 0x00000001)<<31) | (SR>>1);
    return SR & 0x00000001;
}
```

Polinômios primitivos módulo 2

- **Escolha de polinômios primitivos -**

Não há algoritmo eficiente conhecido para geração de polinômios primitivos, mas existem pacotes que testam a primitividade. Polinômios esparsos (com poucos coeficientes $\neq 0$) são mais eficientes para imple-mentações de LFSRs em software, porém mais frágeis para criptografia.

Lista de exemplos de seqüências de captura para períodos máximos (Schneier)			
(7,1)	(46,8,5,3,2,1)	(84,8,7,5,3,1)	(151,3)
(8,4,3,2)	(48,7,5,4,2,1)	(87,13)	(160,5,3,2)
(9,4)	(52,3)	(88,8,5,4,3,1)	(166,10,3,2)
(10,3)	(54,6,5,4,3,2)	(89,51)	(172,2)
(12,6,4,1)	(57,7)	(91,7,6,5,3,2)	(177,8)
(14,5,3,1)	(59,6,5,4,3,1)	(93,2)	(201,14)
(15,1)	(60,1)	(94,6,5,1)	(201,79)
(16,5,3,2)	(63,1)	(96,10,9,6)	(212,105)
(17,3)	(64,3,2,1)	(100,37)	(236,5)
(17,5)	(66,8,6,5,3,2)	(104,11,10,1)	(286,69)
(18,5,2,1)	(68,9)	(107,9,7,4)	(333,2)
(22,1)	(71,6)	(111,49)	(462,73)
(29,2)	(72,10,9,3)	(114,11,2,1)	(607,105)
(32,7,5,3,2,1)	(73,25)	(120,9,6,2)	(1279,418)
(35,2)	(76,5,4,2)	(133,9,8,2)	(2281,1029)
(36,6,5,4,2,1)	(78,7,2,1)	(136,8,3,2)	(3217,576)
(37,5,4,3,2,1)	(79,9)	(144,7,4,2)	(4423,271)
(40,5,4,3)	(80,7,5,3,2,1)	(147,11,4,2)	(9689,84)

Projeto e análise de cifras encadeadas

- **Tipos de ataque específicos a geradores de cifras encadeadas -**

1 - **Complexidade linear:** (Berlenkamp-Massey, 69) Se uma seqüência pode ser gerada por um FSR linear, o comprimento do menor desses LFSR é a medida de complexidade linear da seqüência. Existe algoritmo que reconstrói o LFSR gerador minimal desta seqüência a partir de `2*n` bits consecutivos da seqüência.

2 - **Correlação:** geradores construídos por certas composições de LFSRs podem ser atacados relacionando-se a seqüência de saída com a seqüência de estados internos de elementos do gerador.

3 - **Outros métodos de ataque:** Consistência linear, encontro-no-meio, síndrome linear, aproximação pelo melhor afim, seqüência derivada, criptoanálise diferencial, etc.

- **Técnicas de composição de FSRs -**

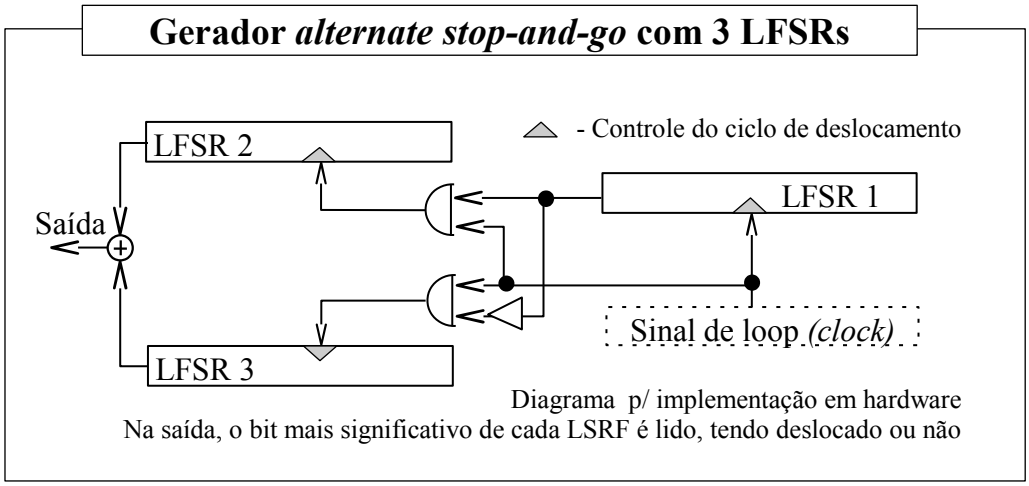
1 - Para-e-segue alternado: *O ciclo de deslocamento de dois FSRs é acionado pelo bit de saída de um outro FSR.*

2 - Cascata de Golleman: *FSR dispostos em cadeia têm seu ciclo de deslocamento controlado pelo FSR anterior.*

3 - Retroalimentação com *carry*: *A função de retroalimentação usa memória de carry da soma dos bits capturados (CFSR)* .

4 - Técnicas obsoletas devido a descobertas de métodos de ataque: *Multiplex, limiar, SFR lineares auto-controlados, para-e-segue não alternado, produto interno, etc.*

- **Exemplo de composição para-e-segue alternado -**



- **Eliminando tendências do gerador -**

  No caso de um gerador apresentar tendência por algum bit na saída, por exemplo, `P(0) = 0.5+e`, pode-se neutralizar esta tendência usando como saída o XOR dos últimos k bits gerados, que terá $P(0) = 0.5+2^{k-1}*e^{k}$

- **Sementes para geradores pseudo-randômicos -**

Existem dispositivos, utilitários ou técnicas disponíveis para gerar seqüências que apresentam características não mensuráveis de aleatoriedade, ideais para uso como sementes de geradores pseudo-randomicos, um dos calcanhares de Aquiles dos protocolos criptográficos. Algumas sugestões.

- Comandos ou posição do mouse;
- Número do setor, hora ou latência de acesso de operações de disco;
- Numero da linha sendo traçada no momento pelo *driver* do monitor;
- Conteúdo do quadro de imagem presente no buffer do driver do monitor;
- Conteúdo das tabelas FAT, tempo de interrupção de threads, tempo entre pacotes de rede,
- etc.

Escolha de algoritmos criptográficos

- **Premissas de confiança quanto às alternativas de origem -**

1 - **Publicado:** *confiança na robustez do algoritmo que resiste à criptoanálise no domínio acadêmico (por pelo menos um ano).*

2 - **Comprado:** *confiança na reputação e lisura do fabricante que projetou e implementou o algoritmo e/ou equipamento.*

3 -.….**Projetado:** *confiança na sua habilidade e perícia próprias para a criptologia, acima das outras alternativas.*

4 - **Escolhido pelo consultor:** *confiança no conhecimento técnico e honestidade de um profissional da área, e na escolha deste.*

5 - **Indicado pelo governo:** *confiança nos propósitos oficiais da padronização proposta se coadunarem com os seus.*

- **Alternativas quanto ao destino de uso -**

1 - **Sigilo e integridade na transmissão de dados:**

- Elo a elo *(link to link) implementação a nível de rede.*
- Ponto a ponto *(end to end) a nível de aplicativo ou sistema.*

2 - **Sigilo e integridade no armazenamento de dados:**

O algoritmo pode funcionar a nível de arquivo ou de volume *(driver).*

3- **Autenticação e gerenciamento de chaves:**

Algoritmos assimétricos são, em geral, mais adequados nestes casos.

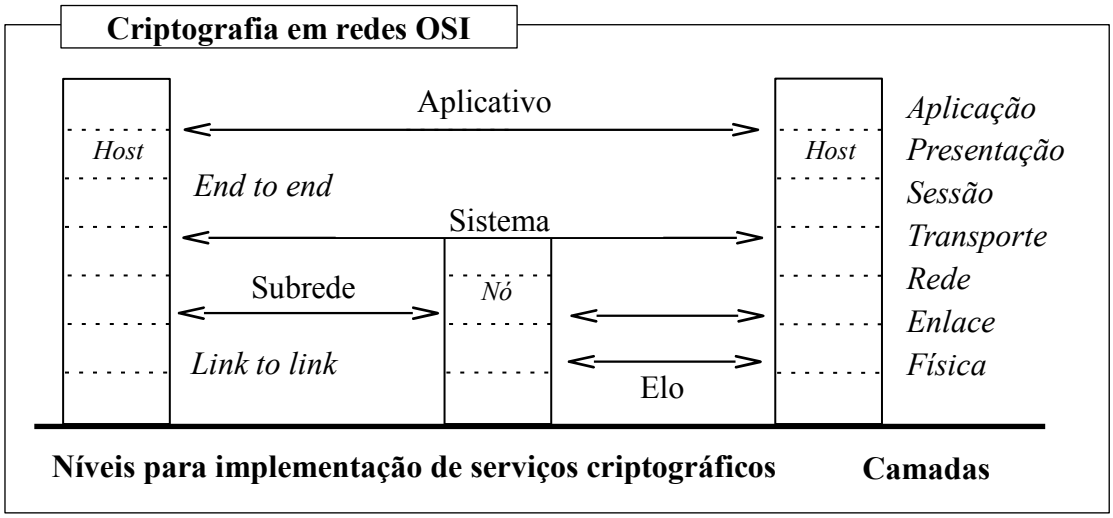
- **Alternativas de plataforma de operação -**

1 – **Software:** Ambiente determina limites ao controle de acesso às chaves.

2 - **Hardware dedicado:** Ambiente ao qual se integra determina limites ao controle de acesso aos serviços criptográficos.

Criptografia na transmissão de dados

- **Possibilidades para implementação no modelo OSI**



- **Comparações entre as alternativas:**

Transmissão	***Link to link***	***End to end***
Proteção nos computadores	• Mensagem exposta no computador de origem e nos nós da rede.	• Mensagem encriptada no computador de origem e nos nós da rede.
Papel do usuário	• Iniciada pelo *host* de origem, invisível ao usuário. • *Host* mantém algoritmo, usado por todos. • Cifra tudo ou nada, podendo ser implementada em hardware.	• Iniciada pelo processo origem, onde o usuário aplica cifra. • Usuários selecionam e mantêm algoritmo caso a caso. • Usuário decide por mensagem, mais fácil por software.
Questões de implementação	• Requer uma chave para cada par de nós, e uma implementação em cada nó. • Cabeçalhos de transmissão encriptados. • Provê autenticação de nós (controle de roteamento)	• Requer uma chave para cada par de usuários, e uma implementação em cada *host.*. • Cabeçalhos de transmissão expostos. • Provê autenticação de usuários (controle de acesso a nível SO)

Criptografia para armazenamento de dados

- **Dificuldades de implementação de cifras p/ armazenagem-**

  1 -.....Vulnerabilidade: *cifras coexistem com dados expostos (no papel, noutra máquina), permitindo ataques de texto pleno conhecido.*

  2 -.....Custo: *em bases de dados, as unidades são muitas vezes menores que o bloco, expandindo o armazenamento.*

  3 -.....Complexidade: *o gerenciamento de chaves deve mapear usuários por direito de acesso a arquivos e campos.*

  4 -.....Gargalo: *dispositivos de I/O são geralmente bem mais rápidos que os algoritmos, requerendo implementação em hardware, ou algoritmos especialmente velozes, ou ambos.*

- **Comparações entre as alternativas:**

Armazenagem	**A nível de arquivo**	**A nível de *driver***
Benefícios	• Mais fácil de implementar, usar e manter. • Transferência fácil de arquivos.	• Arquivos temporários e de trabalho ficam seguros no driver. • Ninguem esquece de encriptar
Problemas para o usuário	• Diferentes senhas para cada arquivo. • Controle de acesso a arquivos geralmente se limita ao utilitário de encriptação.	• Perda da senha mestra compromete todo o disco. • O acesso ao disco se torna mais lento, pois qualquer acesso será cifrado.
Questões de implementação	• Programas que usam arquivos temporários comprometem a segurança. • Implementação pode ser vulnerável mapeando senhas iguais para chaves iguais. • Interface com o usuário pode desestimular o uso de cifragem.	• Extremamente sensível a erros de especificação de programação • Modos de encriptação usuais são inapropriados ou inseguros nesta aplicação (ECB+OFB) • Iteração imprevisiível do driver cifrador com outros drivers de dispositivo.

Escolhas de plataforma

- **Histórico -**

  Desde a invenção em 1920 de máquinas para cifra que usam rotores até recentemente, a implementação de cifras em hardware especializado era prevalente, por ser mais seguro, eficiente e permitir armazenagem interna inviolável de chaves mestras. Esta dominância vem diminuindo pela evolução da criptografia assimétrica e da capacidade dos processadores.

- **Tipos mais comuns de dispositivos de hardware dedicado -**

  1 - **Módulo auto-contido:** usados geralmente por bancos e certifi-cadoras, para autenticação e gerenciamento de chaves criptográficas

  2 - **Caixa-preta para links de transmissão:** cifra implementada para a tecnologia específica de um canal de comunicação (ex: telefonia).

  3 - **Dispositivos de extensão:** usam a arquitetura do PC para interligar hw dedicado à cifragem/gerenciamento de chaves (token, smart-cad,..)

- **Comparações entre as alternativas:**

Plataforma	**Hardware**	**Software**
Benefícios	• Facilidade de instalação, uso transparente e eficiente. • Integridade pode ser obtida com lacre químico inviolável.	• Facilidade de implementação, configuração e atualização. • São portáveis e de baixo custo (algoritmos de domínio público)
Problemas para o usuário	• Informações sobre a origem do algoritmo ou detalhes de implementação são inacessíveis. • O dispositivo é de aplicação específica, requerendo *upgrades* com a troca de equipamentos.	• O armazenamento de chaves e a integridade do código da cifra não são seguros. • O gerenciamento de chaves com essa opção é geralmente de grande complexidade.