SEG INFO 2

12)SEGURANÇA WEB: MODELO

HTTP: esquema + domínio + path + info extra (optional)

Ops: get (sin side efect) para obter, post (reemplazo todos los otros) para crear, put para sustituir, patch para atualizar e delete para borrar.

Estado mantido a travez de cookies

- Server: Db, gestão de sessões,etc.
- Client: Browser html, css, js, eventos.

Browser como pequeno Sistema de virtualização (já q executa código que bem de servers).

Frame: Isolamento entre eles(pag mae funciona incluso se ele falhar).

Dom de js puede mudar a pagina.

Modelos de ataque:

- Browser-Adv-Server: Seg de redes
- Adv controla server: Ataque hacia la pc del cliente
- Adv controla cliente: Ataque hacia el servidor
- Atacante interno/web:
    - 2 tabs, 1 con adv, outro nice,server nice.Browser assegura no interferência entre paginas (malvertising)
    - Adv controla una pagina principal e tem um frame interno nice, server nice, browser idem (pedir recursos a google para ver sim estou logeado)

Modelo de seguranca:

Proceso like Pagina; ficheiro like recursos/cookies; socket/tcp like fetch/http; sub proces like frame

Isolamento: Same Origin Policy(SOP).

Confidencialidad: Nao consego acceder por código a dados de um origen distinto

Integridad: Igual a confidencialidade mas para alterar

- A) DOM: c/ frame tem 1 origem (esquema + domínio + porta) e so acede a dados com = origem
- B) Messages: Frames podem comunicar entre si (programador debe por filtro)
- C) Comunicacao com server
- D) Cookies: envio cookies a server con = origen

C)Frame faz req http a != origenes, tmb expõe seus recursos a != origenes o qual esta bem, mas os dados na resposta não pode ser analizados/modificados programaticamente, so são processados pelo browser aunque embedding expõe algum info (e.g. image login)

- HTML: Crear frame mas não inspeccionar/modificar contido do frame
- JS: obter e executar JS de outra origem no meo contexto mas não inspeccionar/manipular JS de outra origem.

Cross origin resource sharing(cors): Permite a los servicios realaixar politicas SOP

- Pedido simple: A solicita recurso, Browser pede diretamente
- Pedido pre-flighted: Browser faz pedido dummy, server autoriza ou não, se autorizado o browser pde o recurso

D)Cookies:Origem dado por domínio + path + esquema(opcional)

Paginas podem definir cookies para su domínio o de jerarquia superior(excepto suf publico)

Hoje se usa “SameSite” header tal que NONE/STRICT envio cookies so si pedido tem = origem que top level, LAX +é mais relaxado.

Importante usar flag secure cookie para q sea enviada solo por http.Flag HTTP only para que la cookie no sea accesible por javascript

	Do we send the cookie?		
Request to URL	`Set-Cookie: ...; Domain=login.site.com; Path=/;`	`Set-Cookie: ...; Domain=site.com; Path=/;`	`Set-Cookie: ...; Domain=site.com; Path=/my/home;`
`checkout.site.com`	No	Yes	No
`login.site.com`	Yes	Yes	No
`login.site.com/my/home`	Yes	Yes	Yes
`site.com/my`	No	Yes	No

Dominio o path mayor en la url que en la definición de la cookie => envio cookie

Abc.site.com esta abaixo do domínio site.com; a.com/ esta acima de a.com/b/c

Ataque: provoco que cliente envie cookies x a red (http) e eu estou a escuta. E.g. si um server que o cliente acha que es confiável envia JS q hace enviar cookies pela red.

13)ATAQUES WEB 2

Ataques:

- Cross site reuqest forgery (CRSF):
  Server (alvo) nao consege distingir pedido legitimo de falso => para login usar token secreto na form html (browser acede a ele mas JS nao) e en cookies usar flag SameSite=strict.

  Tipos:

  - Site pede recurso noutro sitio, atacante observa a red (session hujacking) => essencial usar https e HTTPONLY flag nas cookies.
  - User autenticado em site x, pedo recurso que causa side Effect (usando um html src url x?transfer…) => o recurso não fica accesible en JS mas o server eecuta
  - Site malicioso nosso browser crea sessão em site alvo para ver histórico de busqueda
- Injeccao de comandos:

  Input malicioso não é validado e causa execução anómala(Shell, db, xml, etc).

  e.g progr lee stdin input e faz system(input), se input= “a.xml;format c:” ataque

  Uso de “--”, “;” , “and” , “or”, “not”. Casos:

  - … where userId = “ + txtUser; input = **“105 or 1=1”**
  - … where name =’ “ + txtName + “ ‘ and pass=’ ” + txtPass; input = “**’ or “”=””**”
  - Igual mas input= **“105 ; drop table Users”**

  Protecciones: Comandos parametrizados (values (?,?)) o bibliotecas ORM os quales sanitizam os paramtros aunque eles mesmos podem ter vulnerabilidades.

- Cross site scripting (XSS):

  Injeccao de codigo no cliente: Atacante faz que site legitimo execute código malicioso para o browser do cliente.

  - Reflected xss: Ocurre instantaneamente
  - Stored xss: Ataque ocure mais tardo (e.g site que faz echo do que recibe)

  Para prevenir debo
  - Filtrar os inputs (blacklist de strings, headers, etc)
  - Content security Policy (CSP): so descargo e executo JS que bem do server de mi white list (definida pro o server).e.g Content-Security-Policy=’SELF’ (outros usos: frame-ancestrors ‘none’, fake paypal atack)
  - Sub-resource Integrity: Fingerprint de JS (HASH)

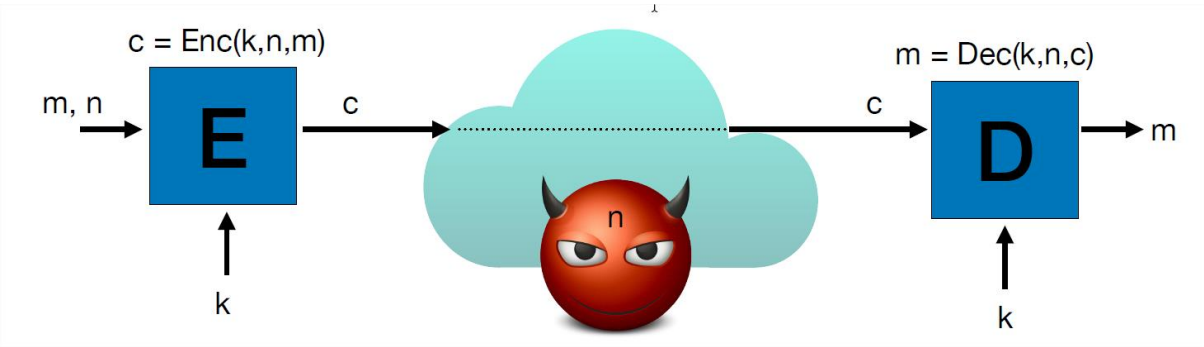
14)CRIPTOGRAFIA 1

Seguranca de info:

En transito online (https) ou offline (email) ou en reposo (disk encryption)

- Confidencialidad: Informacao accesible apenas a emisor e recetor (cifras simétricas e asimétricas,acordos de chaves)
- Autenticidad (e integridad): Receptor tem certeza que os dados vieram de A (assinaturas, acordos de chave, mac).
- Não repudio: emissor não pede negar envio de messages (assinaturas digitais)

Cifras Simetricas:



E,D: Alg públicos e standar; K: chave secreta (128 bits. Pre partilhada) ; n= non repeating

m= texto limpio; c=criptograma

One-time key: nonce é irrelevante, pode ser 0 (e.g mail já que cifra cada message indep)

Many-time key: nonce irrepetible, sequential ou aleatorio. E.g https/tls

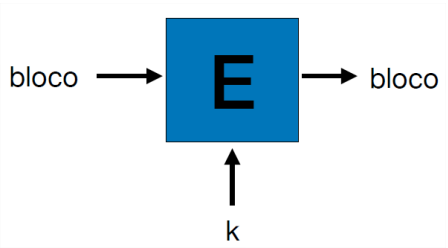
One-time pad: k de tam de m, bits aleatorios, E = M xor K ; D = C xor K .Es bom vs eaversdroop mais (1)exige chave de tamanho de m e (2)so pode ser usada 1 vez.

Soluciones:

1)Gerador pseudo aleatório PRG(K): Dado un K pequeno(8 bit) gero k grande (128 bits)

2)Si fazo C1 xor C2 = M1 xor M2 => uso nounce tal que c1= M xor PRG(K,n1), idem para c2

Cifras de Bloco:

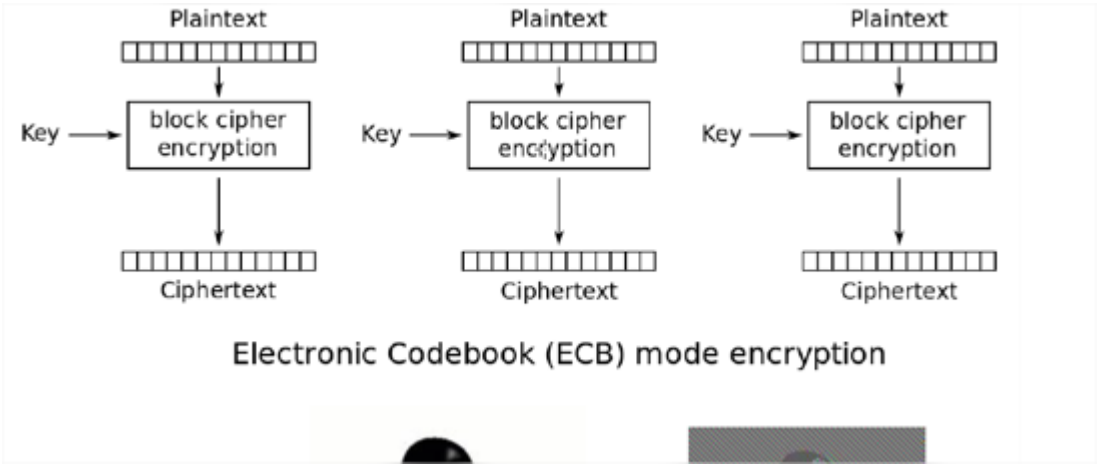


Permiten construir cifras (encryptar informacion) já que para K aleatório e secreto E(K,B) parece aleatiorio incluso escolhendo B:

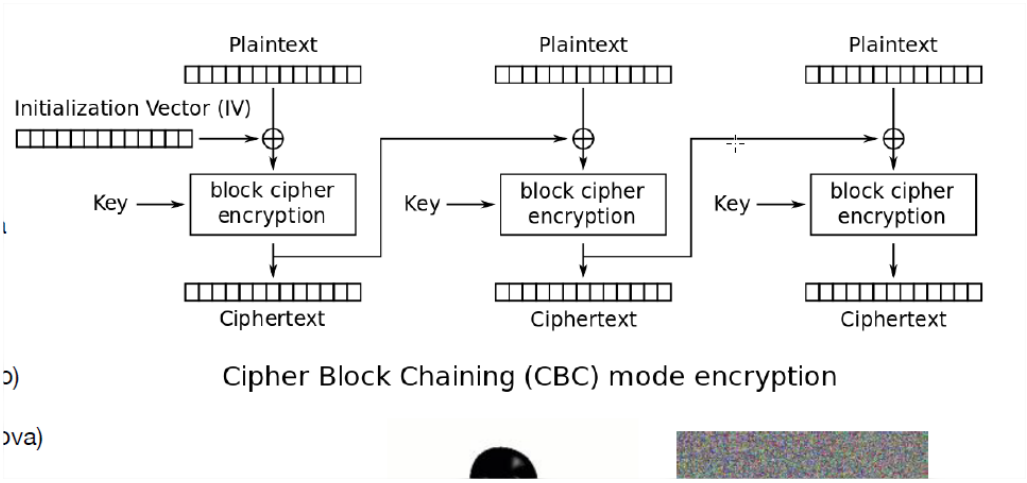
- DES: bloco de 64 bits e chaves de 56 bits (<2000)
- **AES**: bloco de 128 bits e chaves de ate 512 bits. Mais eficiente en HW

Internamente: plain text(p) => R(k1,p) => R(k2,p1)…R(Kn,Pn-1)=> c

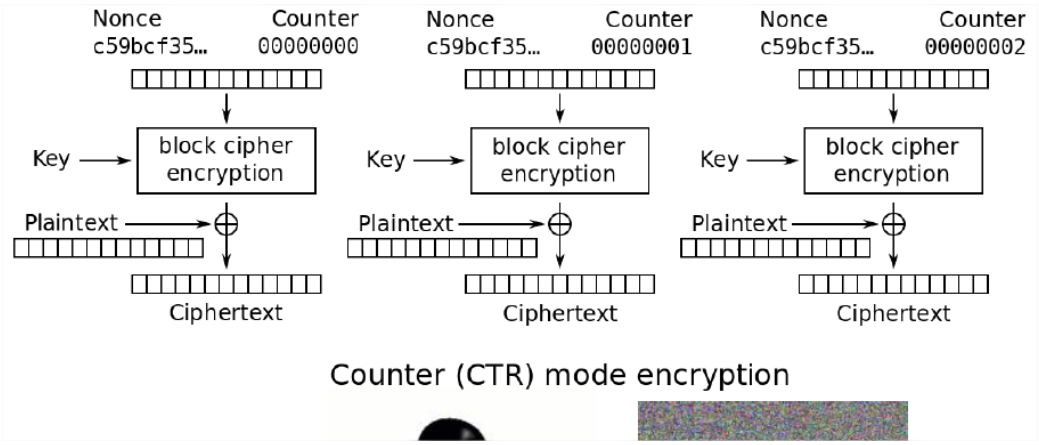
- Electronic code book: Paralelizable mas inseguro ja que bloques iguales dao critpogramas iguais.



- Cipher block chaining mode: Aplica mascara a c/bloco con un cripto anterior



- Counter mode: Similar a one time pad ja que PRG(K,nounce+counter),paralelizable



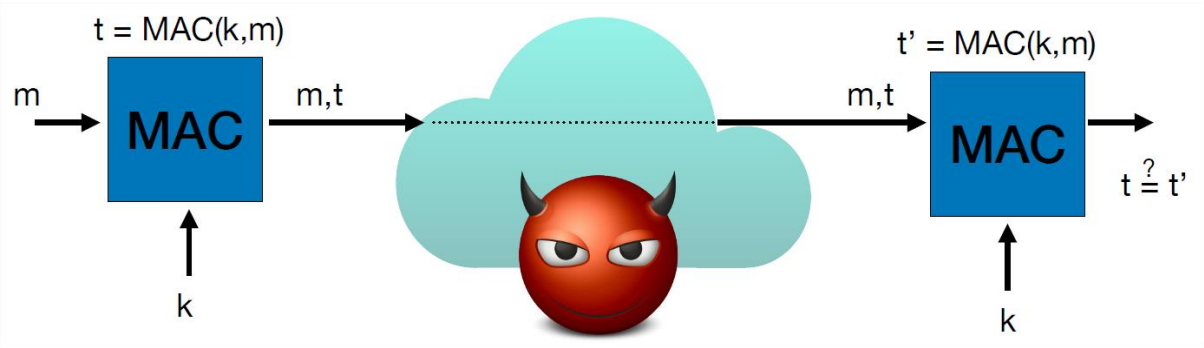
ChaCha20 mejor en SW.Cifra sequencial con nonce, GPR dedicado, similar a counter mode

15)CRIPTOGRAFIA 2

Cifras simétricas nao protegem contra atacantes activos onde eles alteram os criptogramas (mesmo sim eles saber o impacto de esa alteração, eg bit flip) => precisamos integridade e autenticidade (dado um k partilhado por A e B, B acepta msges so de A).

Autenticidad implica integridade já eu si o message e modificado por alguém o emissor já não é o mesmo.

Para eso utilizamos MAC(Message authentication codes) com o sim confidencialidade.



MAC: Alg publico e standar : K: chave secreta(128 bits); m: message(publico) ;t: tag(pequena, 256 bits); MAC es como um chacksum criptográfico

Garantias:

Não protege contra remocao, duplicação e reordenação de messages.So assegura que os msges foram enviados por A alguma vez. Para estos problemas precisamos que M sea único en c/ trasimsao => numero de sequencia n tal que T=MAC(K, N || M).

Trasmito (m,t), receptor tem o mesmo N, tag verifica so se N é igual em ambos lados

Constitucao:

- HashMAC: T = H(okey || H(ikey || m)). **SHA-256** (mac a partir de hash sha-256 com t de 256 bits)
- **Poly 1305**: T = F(m + r) + s. (r,s) es chave secreta de 256 bits, one time mac

Combinacion:

Cifras simétricas para confidencialidade e mac para autenticidade e integridade, si quiero ambas preciso 2 chaves secretas distintas.

- A) Encrypt and mac(ssh): T= mac(m,k2), envio= `c = Enc(k1,m)` `t`
- B) Mac then encrypt(ssl): t= mac(m,k2), envio= `c = Enc(k1,m || t)`
- C) Encrypt then mac(ipsec): t=mac (c,k2), envio= `c = Enc(k1,m)` `t`

Melhor opcao es C já que com B o criptograma pode ser manipulado por estar momentaneamente en memoria antes de validar sua autenticidade mientras que em C so decripto sim o message fora autentico (robusto, seguro e eficiente).

AEAD:

Garante confidecnialidad del message e autenticidade del criptograma e os metadatos(data)

Enc(n,k,m,data) => (c,t)

Dec(n,k,c,t,data) => M se (c,data) for autentico

Implementaciones:

- **AES-GCM**: Mais eficiente en HW
- **CHACHA20 + POLY1305**: mais eficiente en SW (encrypt then mac)

Aleatoriedad:

Idealmente probabilidad de un bit ser 1 ou 0 seria 50% e independiente entre cada bit, na realidade es difícil.

Se utilizan fuentes de entropía (según propiedades fisicas), calculada al inicio e logo atualizadas (resultado aleatorio predecible si no ha muita entropia).

Linux:

Env/random blocking: si no ha suficiente entropía bloquea

Env/urandom: nao bloquea (es preferible pouca entropía que ninguma)

16)CRIPTOGRAFIA 3

Criptografia de chave publica:

Na simétrica temos o problema de gestão de chaves, n sers => n(n-1)/2 chaves

Em sistema fechado precisamos N chaves se usar Key distribution center(kdc) server que almacena 1 chave de longa duração para cada agente e crea chaves de cprta duração para comunicacioes de agentes a<->KDC<->b.O problema é que é um ponto central de falha

- Chaves de curta duração (sessao): Efimeras, danos limitados se comprometida
- Chaves de longa duração : Forte segurança para seu almacenamento

Mas problemas de cripto simétrica:

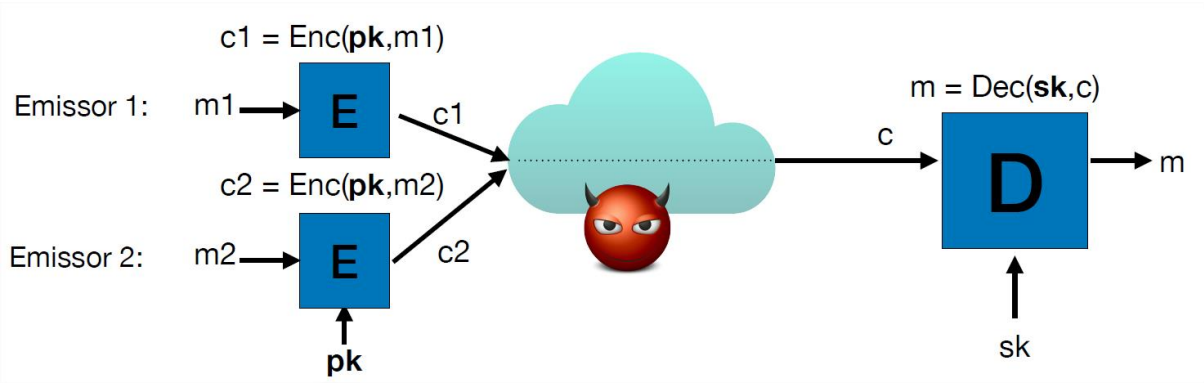
1)A e B confian en distintas autoridades:

- Comunicación assincrona => cifras de chave publica
- Comunicación síncrona => acordos de chaves + assinatura digital

2)Nao repudio: En cifras simétricas y mac ambos partilham as mesmas key =>nao existe

- Sistemas abertos => assinatura digital

Cifras de chave publica: Utilizado para gerir chaves (não cifrar)



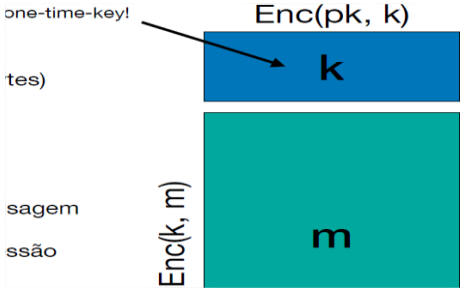
Quem conheca a chave secreta tmb gere e es dono de as chaves publicas

E,D:Algoritmos públicos e standar; pk:chave publica p/ cifrar; sk:chave secreta p/ descifrar

Cifras de chave publica:

+ eficientes q as simétricas, viables apra msges pequenos, o payload é a chave simétrica.

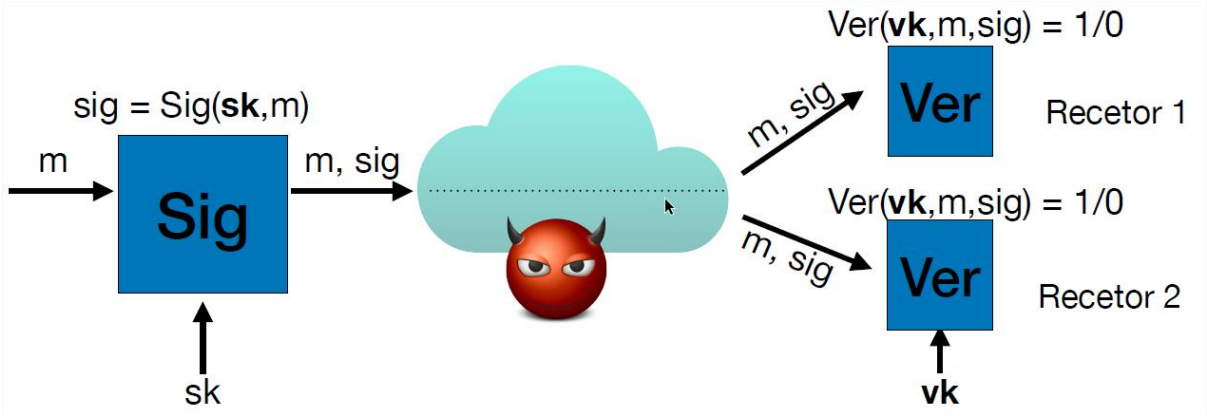
- Emisor gera chave de sessão simétrica e la usa para cifrar o message já que conhece a pk do receptor.
- Receptor recupera chave de sessão K usando su SK, luego o messsage usando K.



Construccion cifras de chave publica (pk,sk):**OAEP**

y=F(pk,x) x=F-1(sk,y) ; permutacao e one-way e.g **RSA**

Assinaturas Digitais: Versao de chave publica de autentiadores tipo MAC



Sig,Ver:Algoritmos públicos e standard ; sk: chave de assinatura; vk: chave (publica) de verificação. O emissor dono de sk, tmb gere e es dono de vk.

e.g: cloud, blockchain, cartao de cidadao

Propiedades:

- Nao falsificavel, nao reutilizavel, não repudiável
- Autoria de documento /acordo contido
- Doc não alterado pos assinatura

Presupuestos:

- Chave não comprometoda
- Alg de assignatura criptográfico seguro
- Chave publica autentica

Contruccion de asignaturas digitais (sk,vk):

- **RSA**: legacy basado en naturales
- **ECDSA**: mas eficiente, basado en curvas eliticas, vk mais pequeñas

Combinacao cifras asimétricas e assinaturas digitais: **envelopes digitales**

Perciso C,I,A: en simétrico obtenho com EncryptThenMac mas não ofrece não repudio já que quem firma puede não conhecer o criptograma => assinar documento so depos de cifrar (e si desejo voltar a signar).

17)CRIPTOGRAFIA 4

Email Seguro:

A conhece PK de B (cifra), B conhece VK de A (autenticacao), quero conf, aut e não rep.

SignThenEncrypt: A signa com sua sk, logo encripta com PK de B.

Como sabe B que o msge recibido era efectivamente para ele? Autenticar metadatos: A tem que por destinatário B antes de firmar

Acordo de chaves: (para gerar chaves de longa duracao)

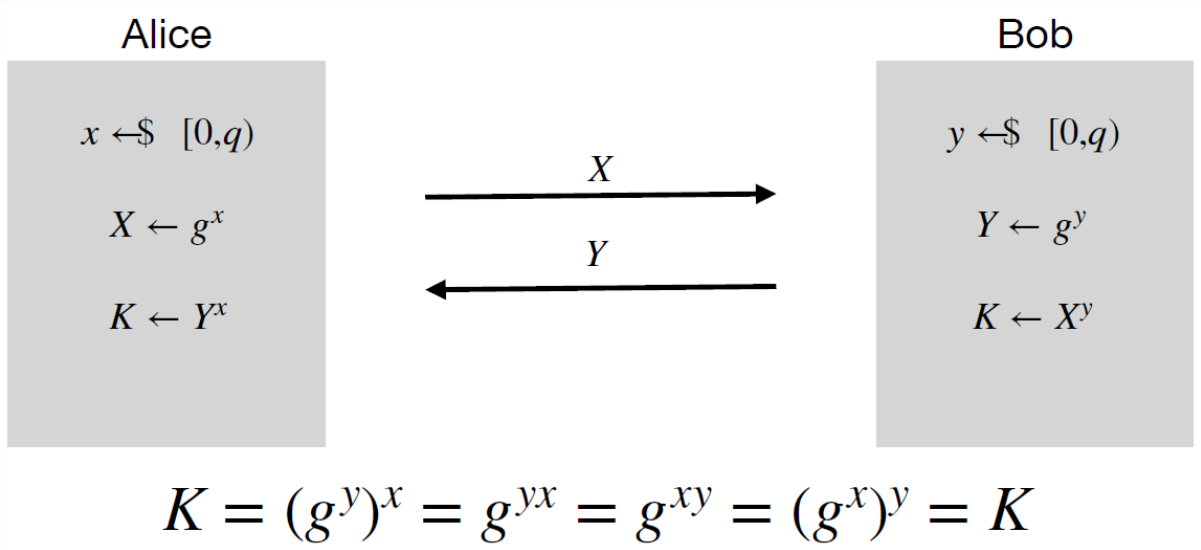
Usa cifras assimétricas ou asignaturas.

A conhee chaves publicas de B para cifrr e verificar assignaturas e vice-versa.

Objetivo: Establecer chave (simétrica?) entre A e B confidencial, autentica e confirmada e com perfect fowards secrecy (compromos de chaves de longa duração não compromete sessões pasadas).

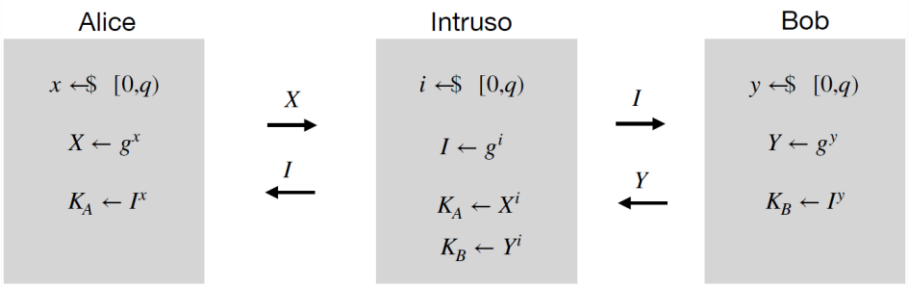
Solucao TLS1.3:

Não utiliza cifras de chave publica para transportar chaves simétricas, usa autenticação (**assinaturas digitais + protocolo diffie helfman** (G,g,o))

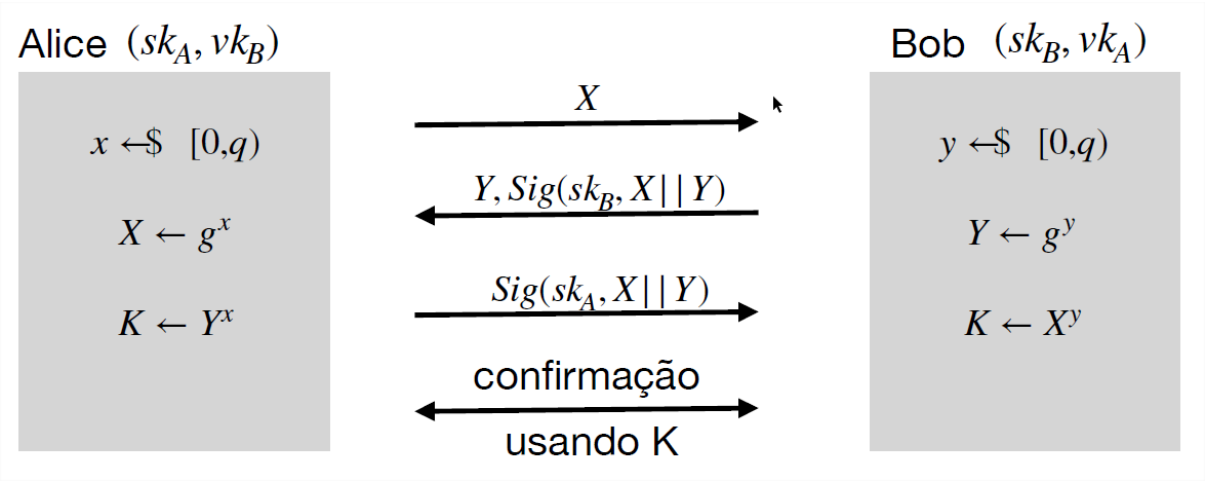


Obtengo k fazendo H(K), requere difícil obter x sabendo X, idem y.

Como nada identifica a A ou B, podo sufrir ataque MIM (o qual é possible se usamos parâmetros públicos troados na red sem saber seu origem, aplica para todo):



Solucao: parâmetros públicos adicionais: chave autenticadas de verificação de assinaturas



X || Y para assegurar que es a mesma sessão.

A autenteicidad das chaves publicas protegem o acordo de chaves o que permite crear um canal seguro já que protege chave simétrica quanto a confidencialidade.

18)PUBLIC KEY INFRAESTRUCUTRE (PKI)

Resolve a necesidad de precisar chaves publicas autenticas.

PKI = PGP/GPG + regulacao

Certificados de chave publica

Objetivo: A envia a B PK x canal seguro. B deve ter certeze que PK pertenece a A

Sol trivial: B pregunta a TTP (Trusted Third Party) se PK é autentico (A demostrou previamente a TPP eso) o qual tem dois problemas:

- 1) Implica ter um canal seguro online 24x7 entre B e TTP
- 2)A e B podem não confiar no mesmo TTP

Soluciones:

1)Certificado de chave publica permite ao TTP(CA certification authority) estar offline

- A prova a CA que posee PL (assinando o pedido ou usando PK fornecida por CA).
- CA verifica os dadoes de A: Identidad, PK, validad, info de CA
- CA assina documento electronico com esa info (certificado)
- A envia o certificado a B por canal inseguro, quem o recebe assinado por CA e o Verifica comenzado por sua info como ID a, pk A, firma de CA e meta info (id pode ser dns),datas de validad.Logo verifica que o CA seja de confianza e obtiene o VK de CA de a list ade CA root suas de confianza e usa esa VK para verificar a assinatura do certificado.

Extenoses: Tem OI, se for critica e não conhecida o certificado é considerado invalido.

As mais importantes são: subject/auth key identifier, basic constraint(root),key usage

PKI: todo o que é preciso para dar garantias de que os pk pertenecem a entidades e as obligaciones de cada parte.

Os certificados circulan dentro de protocolos (ftp,http,etc).

Que garantias da receber um certificado com um CA desconhecido => NADA

Algumas CA já vem com os SO ou browser, outras confiadas pela comunidade, estos root certs são auto assignados com seu SK (subject = issuer = ca name) e contem sua pk

Acredito que esa CA so produc cert verdaderos, ha jerarquías de cas, cadeias, vou validando em cadena ate chegar a uma CA que confio.

- Autoridad de registro (RA):Front end, verifica dados nos CA e verificar a chave privada correspondiente a chave publica do cert
- Autoridad de certificação (CA): backend, high security

Revogar certificados ainda em validez:

- Certification Revocation List (CRL): black list, url no certificiado
- Trusted servie provider list (TSL): White list, apropiado para comunidades pequenas
- Online certificate status protocol (OCSP): blacklist publicada por CA
- Certificate pinning: while list propias, para grandes entidades como google

Certificate policy: Extensao para firmar ou fazer cosas reguladas pela ley.

19)AUTENTICACAO 1

Autenticacao de entidades:

B tem certeza que A participou num passo processal (Gralmente seguido de autorizacao) e que esta a falar com ele agora

1)Solucao criptográfica: Apropiada para entidades nao humanas

B crea desafio aleatório para A, A assina digitalmente e envia a B, quem verifica esa ssinatura/MAC en un tempo limite. Como desafio é imprevisible não se pode fazer replay e as chaves são autenticas então tenho garantia de estar a falar com A.

2)Autenticacao de utilizadores: Apropiada para entidades humanas

User fornece identificação e pede acceso, server pede prova de ID, a qual é fornecida pelo user e o server decide.As probas podem ser únicos o combinaciones de:

- 1)Algo que se sabe: segredo,pass
- 2)Algo que se posee: smartcard
- 3)Algo intrinseco: Biometria

1)Algo que se sabe:

Implica asumir que so 1 pessoa sabe o segredo.Passwords são simples mas exigem canal seguro apra trasmisao e ter certeza que estou a falar com o servidor, de outra forma pode acontecer um MIM obteniendo meu password.A superfície de ataque é grande.

Passwords dificles: Regras por vezes pode ser contra producentes (hard to guess for humans but easy for computers)

Ataques:

Phishing convenciendo a pessoa de cliequear em link, site with names muito similares ao real, sw e hw keylogers, data breaches.

Data Beraches:

Impacto segum como são guardadas as password (não utilizar plain text).O server não precisa saber o pasword, so reconhecerla então pode guardar H(pw) en lugar de pw já que com hash criptográfica é dificl revertir se a pw for difícil aunque pode ser feito por diccionario(hash gigante com pares pk->H(pw)).

8,9 digitos suficiente para frenar ataques de diccionario e forza bruta.

Contra dicionários tmb se pode usar o SALT , que torna hash de cada server distinto já que guardo H(salt, H(R || pw)) sendo o salt aleatório e idealmente != para cada user.

Outra forma de fazer esos ataques mais dificles é usar hash pesadas.

En definitiva es preciso usar multifactor login para evitar ataques, ja que sigue el principio de defesa en profundidad e mais tarde o mais cedo as password vao a ser descobertas.

20)AUTENTICACAO 2

2)Algo que se posee:

Smartcard,rfid,toke,etc. normalmente son second factor (nao proba identidad en si mesmo).

- SmartCards: Cartao que pode processar/almacenar chaves criptográficas (gral + pin)
- OneTime token: Similar a smartcard mas dispositico q mostrar resultados a desafios usando protocol não iterativo: criptografia simétrica, Mac de hora actual, envio de pass e token.
    - Ventages: Defensa en profundidad,códigos mac únicos e imprevisibles
    - Desventages:1 canal so para pass e token (MIM,phishing), server armazena chaves de cifras (não escalable)
- OneTime pass code: idem a token mas com app ou sms. +fácil de gerir mas – independencia entre factores.

3)Algo Intrinseco:Biometria

Caracteristicas físicas, comprotamentales ou ambas.

Ventages: não transferível, usabilidade ideal, garantias fortes

Desventages: Problemas de privacidade, direito a esquecimento

Proceso: Enrollment(Recolha de mostras e extraccao de templates) y Autenticacao (recolha de amostra e match cmo templates)

Autenticaco remota:

- Server recebe so mostra: Confia em alice para obter-as.Toda info almacenada nele
- Server recebe templates:idem confianza . So templates almacenados no server
- Server recebe resultado match. Exige atestação/hw confiável no cliente. Nada es almacenado no server.

Desafios: Precisao, usabilidade, seguridade, frescura de templates, aeitacao, registros mas faciles, balance entre FAR e FRR (fakes + ou -).

Ataques: Intercepcao e usurpação: caracreristica falsa para enganar ao sensor (e.g replay atack), se podem mitigar com maior precisão ou multifactor.

Autenticacao e sessões web:

Sessao: seq de ped/resp curta ou longa em sites/apps onde autentico 1 vez, uso n.

Prehistoria: http auth, c/pasta com hash de password no server: confuso, não multiple users

Token de sessão: Na autenticação server crea token que fica guardado no cliente e é devolvido en os pedidos futuros.Guardado em cookies, link, form hidden.(hoje dia todos).

Ataques:

- Roubo de token (XSS,MIM,eaversdrooping, falha em logout) => mitigação ligar token a maquina (ip)
- Token fixation: Atacante inicia sessão e recebe o token,convence user de fazer login com mesmo token aumentando os priilegios do token => Siempre usar distintos tokens en não elever privilégios de tokens

21)SEGURANÇA DE REDES 1

Protocolos y capa OSI: todos usan ipv4 ou ipv6 y Camada inferior encapsula info de camada superior.

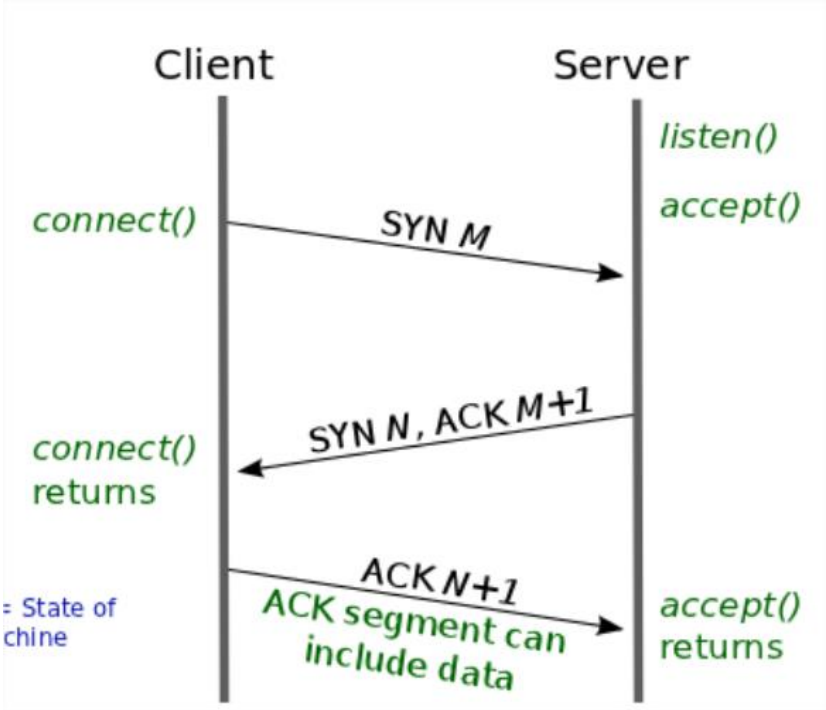
Al recibir un pacote com source address nao ha autenticacao => adv pode injectar pacotes

Tabela de enderecos mac constituída com ARP.

Req: meu mac + ip deseada ; Resp: meu mac + tu mac + meu ip

Router se basa em tavela de routing e em BGP => propagasao por gossip

TCP triple handshake:Eleccion de nro de seq es importante para segurança



Em tcp flag FIN finaliza a conexão, Flag RST a finaliza com error. UDP = TCP en segurança

CIA:

Ataques em todo nível: físico com ligação directa, dispositivos manipulados o infraestructura

Adversarios: eversdrooper, off path (so insere pac), on path (olha e insere),MIM

Ataques:

Camada Fisica:

- Wiretapping: Adv liga equipamento ao medio da comunicação e escuta/modifica info
- Everdrooping: Info não cifrada => sirve.Basta acceso ao medio de comunicação(wifi)

Camada Logica:

- Mac flooding: Enviar muchos msges com mac != .Switch passa a ser hub(broadcast)
- Mac spoofing: Robo mac de 1 ip que conheço configurnado nuestra placa de red com esa ip já que não há autenticação

Camada de red:

Envio pacotes a quem quizer já que não há autenticação

- Scannig: Mando a toda a red e vejo quem responde (alvos potenciales)
- DoS: Sobrecargo alvo com messages
- ARP Posoning/spoofing: Respondo ARP com mi mac dizendo que yo soy que posee a ip anunciada => MIM(leo o message e logo envio a ip correto)
- Hijacking de routing: IRDP para descubrir routers. Adv faz que mac locales o usem como router já que infor de reoter não e autenticada
- Roughe DHCP: Falso server envia resposta DHCP para configurar su gateway(MIM)
- DNS spoofing: server dns de atacante da ip falsas aos users (primeiro debo convencer user de usar meu server como server dns).
- DNS(cache) poisoning_ bombardeo server dns com info falsa.

22)SEGURANÇA DE REDES 2

Camada de transporte (TCP)

Não há autenticação => não há forma de verificar que o message bem de quem dice vir

Ataques:

- Simple: enviar cabeçalhos apropriados e rst para terminar ligação.Requere gran control de a infraestructura (e.g china).
- Spoofing as cegas (off path): envio syn e adivinho nro de seq (probable quando ele esta basado no relog) => mitigação usar nros de seq aleatórios
- TCPSession Hijacking (on path): tomar conta de ligação logo de que A e B já fizeram hand shaking e autenticação.
    - As fases incluem tracking, des-sincronizacao e injeccao.
    - Es fácil e eficaz se não há criptografia.
    - Faz bypass a autenticação e enganha router/proxies
    - Mas é + difícil que spoofing:Requere timing e gran conhecimento de sessão
    - Funcionamento: Uso sniffing para obter info da sessão e logo envio nro de seq mayor forzando uma resincronizacao => janela de tempo para atacar

UDP es similar a TCP mais não é necessário ser mais rápido na resp e precisa menos info

Soluciones:

Criptografia entre os endpoints en Red(ipsec), Transporte (TLS) e apps (whatsapp)

Incluso con cripto ha ataques de infraestructura, metadatos, DoS.

Defesas:

Desligar todos os servicios desnecessários? Dificil e insuficiente

Sol típica: Firewall + nat + proxies + network intrusion detection system

Firewalls:

- Firewwall host: app rules
- Firewall red: Intercepta comunicacion de e para o exterior

Hacen filtragem de pacotes maliciosos (!= proxies q trabalham a nível de app)

Politicas de control de acceso: Flexible de dentro para fora e conservador no outro sentido

O que não é apanhado por eles va para default allow (permito todo menos certos problemas conhecidos) ou default deny(permito so alguns accesos especificos) => foward ou drop

Filtragem sem estado: olho a cada pacote individualmente. Menos trabalho e + permissivo

Filtragem com estado: olho a cada pacote segum contexto. Menos trabalho e + inteligência

Nat:

Ventagem: reduce exposição ao exterior e as ligações começadas de fora são descartadas

Desventagem: Perturba outros protocolos e é vulnerável a nat flooding

Proxies:

MIM do bem, direccionados a apps

Deteccion/prevensio de intrusões:

IDS/IPS em comuniacoes: Identificar ataques por pacotes. Host ou Net.

NIDS: Olho tablea de ligações e procuro padroes

- Ventages: Não configuro cada pc individualmente
- Desventages:exigente e menos preciso (menos falsos - => mais falsos +)

HIDS:Defensa en prof en maquinas especificas

- Ventages:Información ja decriptada
- Desventages: costo => so uso em algunos pcs (servers)

Analisis de logs:

Baratos, offline, apenas reativo e o atacante pode mudar logs. São utilez para melhorar as politicas dos firewalls

Pen-Testing:

Simular ataque (já sea com ferramenta ou com consultores) => proativo e optimizacao mas caro e com possible consequencas como DoS

Honey-Pots:

Trampa para hackers (e.g. queso para raton)

23)TRANSPORT LAYER SECURITY (TLS)

Es un protocolo de transporte.Mesmo sendo seguro transmite mucha meta informacao

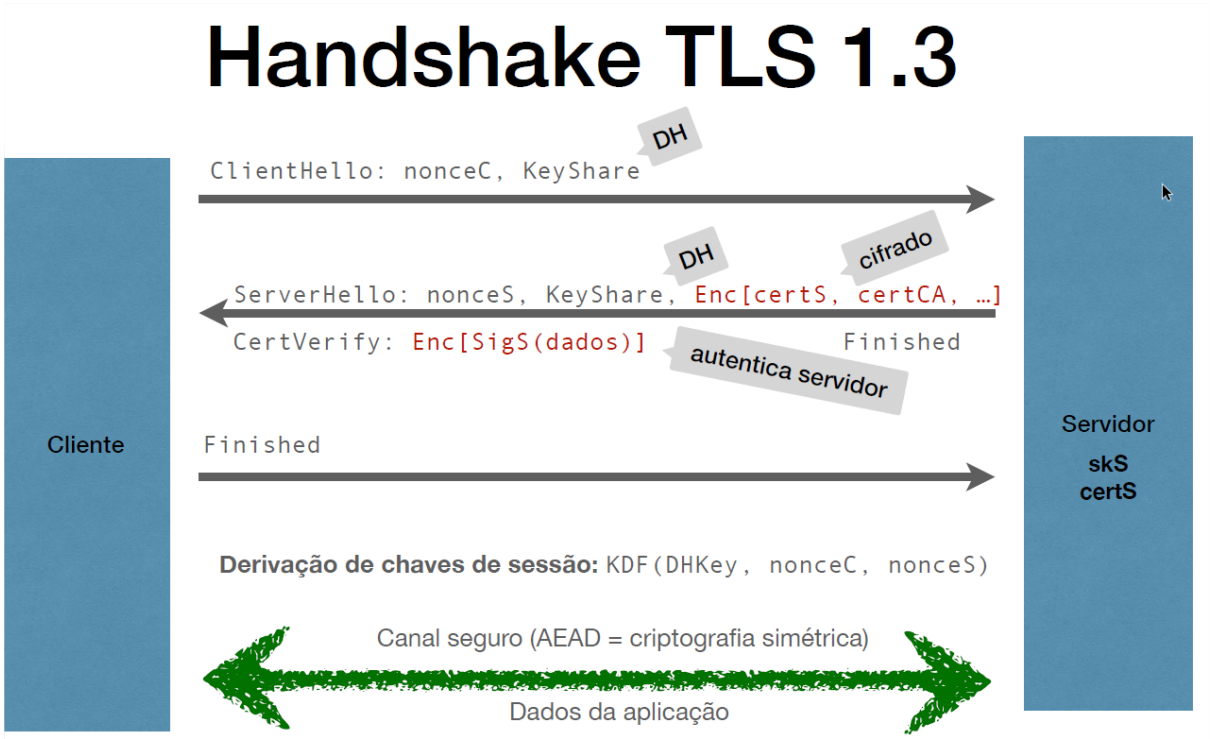
Modelo de seguranca: Atacante controla infraestructura

Handshake TLS1.3: Diffie- Hellman autenticado

Historicamente: Cliente envia chave de sessão ao server (RSA), quando el server la usa se verifica su autenticacion implicitamente

Hoje:DH autenticado com curvas elípticas (ECDSA?) por sua eficiência e por dar perfect foward secrecy(chaves de logna duração para assinatura e não para transporte de chaves de sessao => chaves de longa duração não compromete acordos de chaves pasados)

TLS implica PKI já que server envia cert a cliente que se quere comunicar com ele, chave publica no cert utilizada para validar assinatura digital no protocolo DH



Existe uma optimizacao que reutiliza chaves mas permite ataques por repetição

Integracao TLS/HTP:

Http como payload do TLS => problema já que web proxy precisa destrino HTTP => solução: Nome de domínio en TLS e virtual host: mesmo ip com muchos DNS => incluir nome de domínio de server no cliente hello

Usar sempre HTTPS?Antes no se podia por performance, hoje se puede

TLS/HTTPS en browser: Redireccionar http para https mas permite MIM ssl strip atack (e.g adversario faz https com paypal e http com nos) => solucion usar flag na cache de browser strict transport security forzando a ligações futuras ser https.

MIM evitable se conseguimos autenticar chave publica do servidor.

Não mixar http com https: não buscar scripts/images com http já que MIM pode manipular-o