anonymni platby

problém kreditních karet spočívá v sledovatelnosti toku peněz. Hledáme protokol pro tvorbu autentizovaných ale nesledovatelných zpráv.

1. Zákazník *A* připraví 100 anonymních příkazů k platbě na stejnou částku. Každý z příkazů vypadá následovně:

Množství:	Kč 1,-
Jednozn. řetězec:	X
Identifikační řetězce:	$\left(I_{1_L}, I_{1_R}\right)$
	$I_2 = \left(I_{2_L}, I_{2_R}\right)$
	...
	$I_{100} = \left(I_{100_L}, I_{100_R}\right)$

Každý příkaz obsahuje 100 různých párů identifikačních řetězců, každý vzniklý z řetězce obsahujícího úplnou identifikaci zákazníka vhodným algoritmem pro secrets splitting. Např. každý pár může být tvořen párem paketů podobně jako v protokolu pro bit commitment tak, aby se dalo kontrolovat rozkrytí paketu. Známe-li obě poloviny, můžeme sestavit původní řetězec.

2. *A* "zaslepí" všechny příkazy protokolem pro podpisy naslepo a odešle je do banky *B*.
3. *B* požádá *A* o "odslepení" náhodně zvolených 99 příkazů a rozkrytí všech identifikačních řetězců
4. Je-li vše v pořádku, banka *B* podpisem potvrdí zbylý příkaz a vrátí jej *A*.
5. *A* "odslepí" potvrzený příkaz a předá jej obchodníkovi.
6. Obchodník ověří podpis banky a tím legitimnost příkazu.
7. Obchodník požádá *A* náhodně o rozkrytí jedné poloviny každého páru identifikačních řetězců.
8. *A* splní požadavek.
9. *B* ověří svůj podpis a zjistí, zda již nepřijala příkaz se stejným *Jednozn. řetězcem*. Pokud je vše v pořádku, vyplatí *B* peníze a příkaz archivuje.
10. Pokud příkaz se stejným *Jednozn. řetězcem* již banka přijala provede zkoumání rozkrytých identifikačních řetězců. Jsou-li v obou případech stejný, podvádí obchodník, jinak podvádí *A*.

Protokol zajišťuje, že obchodník ani *A* nemohou podvádět. Musí však věřit bance, že s jejich účty nakládá korektně. Na druhou stranu pokud se oba chovají korektně, banka nemá k dispozici žádnou informaci o tom, jak nakládají se svými finančními prostředky.

Blowfish

opět Feistelova šifra, délka bloku 64 bitů, proměnná délka klíče až 448 bitů

Subklíče

předpočítávají se před každým šifrováním

- *P-pole* = 32 bitové klíče
  $P_1, P_2, \ldots P_{18}$
- pole *S-boxů*, každý 256 32-bitových položek
  $S_{1,0}, S_{1,1}, \ldots S_{1,255}$
  $S_{2,0}, S_{2,1}, \ldots S_{2,255}$
  $S_{3,0}, S_{3,1}, \ldots S_{3,255}$
  $S_{4,0}, S_{4,1}, \ldots S_{4,255}$

Nereverzibilní funkce F:

$$\mathrm{F}(x_L) = \left(\left(S_{1,a} + S_{2,b} \bmod 2^{32}\right) \mathrm{xor}\, S_{3,c}\right) + S_{4,d} \bmod 2^{32}$$

$$x_L = a|b|c|d$$

Generování podklíčů

1. Inicializujeme *P-pole* a všechny *S-boxy* pevným řetězcem
2. **xor** P_1 s prvními 32 bity klíče, **xor** P_2 s dalšími 32 bity klíče atd.
3. Zašifrujeme nulový řetězec
4. P_1 a P_2 nahradíme výstupem předchozího kroku
5. zašifrujeme výstup kroku 3.
6. P_3 a P_4 nahradíme výstupem předchozího kroku
7. stejně pro ostatní položky *P-pole* a všechny *S-boxy*

Algoritmus provádí 16 cyklů nad vstupem délky 64 bitů. Pro účely analýzy navrženy jeho zmenšené varianty (MiniBlowfish) pracující nad vstupem 32 popřípadě 16 bitů.

casove znamky

nejjednodušší metodou je zasílat kopie zpráv důvěryhodnému arbitrovi, problémy s množstvím uchovávaných dat lze vyřešit použitím hašovacích funkcí

Spojované (linked) známky

- aby odesílatel (adresát) společně s arbitrem nemohli podvádět

1. Odesílatel *S* zašle arbitrovi *A* hashkód zprávy H_n.
2. *A* vrátí odesílateli

$$T_n = S_K\left(n, S, H_n, Tm_n; ID_{n-1}, H_{n-1}, T_{n-1}, \mathbf{H}\left(ID_{n-1}, H_{n-1}, T_{n-1}\right)\right)$$

   kde n je pořadí zprávy, Tm_n čas podpisu zprávy, Id_{n-1} … jsou informace o předešlé zprávě, kterou arbitr vyřizoval
3. po vyřízení následující zprávy arbitr zašle odesílateli identifikaci následujícího odesílatele

Chce-li někdo ověřit časovou známku zprávy, kontaktuje odesílatele Id_{n-1} a Id_{n+1} a pomocí nich ověřit platnost T_n.

Pro zvýšení bezpečnosti je možné připojit informace o více předchozích zprávách a držet seznam stejného počtu následujících odesílatelů

Další možností je tento protokol:

1. H_n užijeme jako vstup vhodného generátoru náhodných čísel, m nejbližších výsledků bereme jako identifikace uživatelů $V_1, \ldots, V_m$
2. všem V_i zašle *S* hashkód H_n
3. V_i připojí informaci o času, celý balíček elektronicky podepíše a vrátí *S*
4. *S* uschová všechny odpovědi jakožto časovou známku

použití generátoru náhodných čísel v 1. kroku zajišťuje, že *S* nemůže podvádět, neb neví předem, kdo bude jeho zprávu podepisovat, navíc okruh verifikátorů se bude pokaždé měnit

certifikaty

Služby správy klíčů:

…
Tvorba certifikátů - certifikáty používány pro účely autentizace. Autorita provádějící certifikaci doplněním certifikátu k danému objektu prohlásí tento za “pravý”.
on-line / off line

choosen x attack

Útoky lze rozdělit dle celé řady kriterií. Časté je dělení

Znalost zašifrovaného textu (ciphertext only att.)

útočník má k dispozici pouze zachycený zašifrovaný text, dále může využívat apriorních informací - pracuje tedy jen se statistickými rozbory, distribucí, pravděpodobností.
Systém, který není odolný vůči tomuto útoku nelze označit za bezpečný.

Znalost otevřeného textu (known plaintext att.)

předpokládá se, že útočník má k dispozici pár otevřený text + odpovídající šifra

Pravděpodobný text (probably plaintext att.)

útočník na základě okolností odeslání zprávy může učinit částečný odhad obsahu zprávy

Zvolený otevřený text (choosen plaintext att.)

útočník může získat k libovolnému otevřenému textu odpovídající šifru, velmi používaný útok, vhodný i proti statistickým databázím

Zvolená šifra (choosen ciphertext att.)

používá se v případě, že útočník může šifrovacím algoritmem zašifrovat velké množství zpráv, aby našel plaintext odpovídající zvolené šifře.

Digitální podpisy

- key-only attack – zná pouze veřejný klíč oběti
- known signature attack – dtto + má k dispozici pár (zpráva-podpis)
- chosen signature attack – útočník si může vybrat zprávy, které si nechá podepsat

co je to s-box

Substitution box

In cryptography, a substitution box (or S-box) is a basic component of symmetric key algorithms. In block ciphers, they are typically used to obscure the relationship between the plaintext and the ciphertext — Shannon's property of confusion. In many cases, the S-boxes are carefully chosen to resist cryptanalysis.

In general, an S-box takes some number of input bits, m, and transforms them into some number of output bits, n: an m×n S-box, implemented as a lookup table. Fixed tables are normally used, as in the Data Encryption Standard (DES), but in some ciphers the tables are generated dynamically from the key; e.g. the Blowfish and the Twofish encryption algorithms.

co jsou to inicializacni vektory

víme

Inicializace

nejprve se doplní zpráva do násobku 512: doplní se 1, potom tolik nul, aby zpráva byla o 64 krtatší než nejbližší vyšší násobek 512 a na konec do 64 bitů zapsaná původní délka zprávy

co jsou to mekke (weak) klice

Fixed point attack

Hledáme H_{i-1} a X_i tak aby

$$\mathbf{f}\left(X_i, H_{i-1}\right) = H_{i-1}$$

pokud zřetězující proměnná nabyde hodnoty H_{i-1} , můžeme vložit libv. množství bloků X_i. Útok je reálně možný pouze pokud můžeme manipulovat s hodnotou inicializačního vektoru, nebo pokud **f** má velké množství pevných bodů.
Snadnou obranou je ke zprávě přidat její délku.
Hledání pevných bodů lze používat i při analýze šifrovacích algorimů. Šifrovací algoritmus lze brát jako náhodnou permutaci a tedy je pravděpodobné, že pevné body lze najít. V DES pro skupinu weak a semiweak klíčů existuje celkem 2^{33} pevných bodů.

DES

Vyvinula firma IBM na zakázku NBS počátkem 70. let. Původní název DEA, v USA DEA1. Jako standard přijat 23. 11. 1976
Dodnes používán v komerční sféře, pro vojenské účely není certifikován ani pro ochranu neklasifikovaných informací. Patrně nejrozsáhleji používaný šifrovací algoritmus všech dob.
Norma ANSI X3.92
šifruje 64-bitové bloky otevřeného textu na 64-bitové výstupní bloky, délka klíče 64 bitů

Požadavky zadavatele

1. Alg. musí poskytovat vysoký stupeň ochrany
2. Musí být formálně popsatelný a snadno pochopitelný
3. Bezpečnost algoritmu nesmí záviset na znalosti či neznalosti samotného algoritmu.
4. Musí být dostupný pro nejširší veřejnost.
5. Musí být použitelný v nejrůznějších aplikacích.
6. Musí být efektivní.
7. Musí být ověřitelný.
8. Musí být exportovatelný.

Analýza

- úvodní permutace nemá prakticky žádný vliv
- příliš krátký klíč, navíc efektivně pouze 56-bitový
- komplementárnost - $C = \mathbf{E}\left(K, P\right) \Leftrightarrow \neg C = \neg\, \mathbf{E}\left(\neg K, \neg P\right)$
- existence slabých (weak) klíčů ($\mathbf{E}(K) = \mathbf{D}(K)$) a poloslabých (semiweak) klíčů ($\mathbf{E}(K_1)\ \mathbf{E}(K_2)) = Id$.
- nevhodný návrh S-boxů

Možné způsoby zvýšení bezpečnosti

1. vícenásobné šifrování - nestačí dvojnásobné, ke skutečnému zvýšení bezpečnosti nutno šifrovat

$$C = \mathbf{E}\left(K_1\right)\mathbf{D}\left(K_2\right)\mathbf{E}\left(K_1\right)$$

2. zvětšení délky hesla na 768 bitů - nepříliš účinné

Poučení z DESu

- paralelní aplikace malých S-Boxů je nevhodná pro SW implementaci
- příliš malý klíč
- permutace a permutační výběry těžko zvládnutelné v SW
- Feistelova síť principiálně funguje a vykazuje dobrou odolnost vůči analýze
- nové algoritmy by měly být vhodné pro SW implementaci na běžném HW (výběr operací)
- je třeba zvýšit odolnost vůči masivně paralelnímu útoku (drahá key schedule)

- rozhodně delší klíče, případně blok
- vysokou odolnost vykazují širší S-boxy a na klíči závislé interní struktury algoritmu
- nahrazení paralelních operací, které jsou neefektivní a mají omezený lavinový efekt sekvenčními operacemi

DH-kryptosystem

Autory jsou Diffie a Hellman
Metoda je založena na obtížnosti počítání logaritmů nad GF(p)

1. Uzel A vypočítá

$$Y_a = \alpha^{X_a} \bmod p$$

   a pošle je druhé straně.

X_a - náhodně zvolené číslo, α - některý z generátorů GF(p).

2. Obdobnou akci provede i uzel B Y_b.
3. Obě strany spočtou

$$K_{ab} = \alpha^{X_a X_b} \bmod p$$

   Případný útočník by musel K_{ab} spočítat pouze s použitím Y_a nebo Y_b což nejde rychleji, než spočítat logaritmus jednoho z čísel X_a nebo X_b.

Má-li *p* okolo 300 cifer, A i B potřebují řádově tisíc operací, potenciální útočník zhruba 10^{30} operací.
Dnes minimálně 1024 bitů (dále viz RSA)

digitalni volby

- volit smí pouze oprávnění voliči
- každý smí hlasovat nejvýše jednou
- nikdo nesmí vědět, kdo jak volil
- nikdo nesmí měnit volbu jiných
- každý hlas musí být započítán
- je zjistitelné, kdo volil

Poslední podmínka již není nutná.

Protokol se dvěma centrálními autoritami

používá registrační autoritu *RA* provádějící registraci voličů a sčítací autority *SA*, která sčítá hlasovací lístky a zveřejňuje výsledky voleb

1. všichni voliči zašlou *RA* žádost o validační číslo
2. *RA* zašle každému voliči náhodně zvolené validační číslo *L*, zároveň si ponechá seznam, kdo jaké číslo dostal
3. *RA* zašle seznam validačních čísel *SA*
4. každý z voličů si náhodně vybere svoje identifikační číslo *Id* a *SA* zašle zprávu

$$(L, Id, v)$$

   kde *v* je volba
5. *SA* porovná *L* se seznamem validačních čísel z kroku 3. odpovídající číslo škrtne a voličovo *Id* přidá do seznamu asociovaného s voleným kandidátem
6. po skončení voleb *SA* zveřejní výsledky voleb a seznamy identifikačních čísel spojené se jmény kandidátů

Každý z voličů si může ověřit zda byl jeho hlas správně započítán, *RA* zjistí případné falešné hlasy.

SA však může registrovat neoprávněné voliče, případně některé voliče vícekrát. V případě že budou *RA* a *SA* spolupracovat, hrozí nebezpečí porušení anonymity voličů.

Protokol bez centrální autority

voliči *A, B, C, D* se rozhodují ano či ne

1. všichni voliči zváží své rozhodnutí a
   (a) ke své volbě připojí náhodný řetězec
   (b) zašifrují výsledek kroku (a) veřejným klíčem voliče *D*
   (c) zašifrují výsledek kroku (b) veřejným klíčem voliče C
   (d) zašifrují výsledek kroku (c) veřejným klíčem voliče *B*
   (e) zašifrují výsledek kroku (d) veřejným klíčem voliče *A*
   (f) připojí k výsledku kroku (e) nový náhodný řetězec, jehož hodnotu uchovají, a celé to zašifrují veřejným klíčem voliče *D*
   (g) připojí k výsledku kroku (f) nový náhodný řetězec, jehož hodnotu uchovají, a celou zprávu zašifrují veřejným klíčem voliče *C*.
   (h) připojí k výsledku kroku (g) nový náhodný řetězec, jehož hodnotu uchovají, a celou zprávu zašifrují veřejným klíčem voliče *B*.
   (i) připojí k výsledku kroku (h) nový náhodný řetězec, jehož hodnotu uchovají, a celou zprávu zašifrují veřejným klíčem voliče *A*.

   vcelku tedy

$$E_A\left(R_5, E_B\left(R_4, E_C\left(R_3, E_D\left(R_2, E_A\left(E_B\left(E_C\left(E_D\left(v, R_1\right)\right)\right)\right)\right)\right)\right)\right)$$

   kde R_i je náhodný řetězec a v voličovo rozhodnutí
2. všichni voliči pošlou výsledky svých výpočtů voliči *A*

…

DSA

The Digital Signature Algorithm (DSA)

Key generation

Choose a 160-bit prime q.
Choose an L-bit prime p, such that p=qz+1 for some integer z and such that $512 \le L \le 1024$ and L is divisible by 64.
L should only assume the value 1024, and the forthcoming FIPS 186-3 (described, e.g., in SP 800-57) uses SHA-224/256/384/512 as a hash function, q of size 224, 256, 384, and 512 bits, with L equal to 2048, 3072, 7680, and 15360, respectively.
Choose h, where $1 < h < p - 1$ such that $g = hz \bmod p > 1$.
Choose x by some random method, where $0 < x < q$.
Calculate $y = gx \bmod p$.
Public key is (p, q, g, y). Private key is x.
Note that (p, q, g) can be shared between different users of the system, if desired.

Signing

Generate a random per message value k where $0 < k < q$ (this is known as a nonce)
Calculate $r = (gk \bmod p) \bmod q$
Calculate s = (k-1(SHA-1(m) + x*r)) mod q, where SHA-1(m) is the SHA-1 hash function applied to the message m
Recalculate the signature in the unlikely case that r=0 or s=0
The signature is (r,s)

Verifying

Reject the signature if either 0<r<q or 0<s<q is not satisfied.
Calculate w = (s)-1 mod q

Calculate u1 = (SHA-1(m)*w) mod q
Calculate u2 = (r*w) mod q
Calculate v = ((gu1*yu2) mod p) mod q
The signature is valid if v = r
DSA is similar to the ElGamal signature scheme.

dukazy s nulovymi znalostmi

Důkazy s nulovou informací (zero-knowledge proofs)

- dokazovatel (prover) nesmí podvádět, pokud důkaz nezná, jeho šance přesvědčit ověřovatele je mizivá
- ověřovatel rovněž nesmí podvádět, o samotném důkazu smí zjistit pouze to, že jej dokazovatel zná. Zvláště nesmí být schopen celý důkaz rekonstruovat a sám provést.
- ověřovatel se nesmí dozvědět nic, co by nebyl schopen zjistit bez pomoci dokazovatele.

není-li splněna poslední podmínka, mluvíme o *důkazech s minimálním vyzrazením* (minimum-disclosure proofs)

jeden z možných důkazů založen na problematice Hamiltonovských kružnic v grafu

1. Nechť *A* zná Hamiltonovskou kružnici v grafu *G*
2. *A* provede náhodnou permutaci *G*. Původní graf a vzniklý *H* jsou izomorfní.
3. Kopie grafu *H* je zaslán entitě *B*
4. Ověřovatel *B* položí dokazovateli *A* jednu z následujících otázek:
   a) dokázat, že *G* a *H* jsou izomorfní
   b) ukázat Hamiltonovskou kružnici v grafu *H*
5. opakováním kroků 1. až 4. lze docílit potřebné jistoty

el-gamal

El Gamal kryptosystém

založen na obtížnosti výpočtu diskrétního logaritmu nad okruhem

Konstrukce kryptosystému

Společný modul *q*, dále je zvoleno číslo *g* co nejvyššího řádu (nejlépe generátor). Každý účastník *i* si zvolí tajný klíč y_i a vypočítá veřejný klíč $g^{y_i} \bmod q$

Šifrování

nechť uživatel *A* posílá zprávu *P* (< q) uživateli *B*
náhodně vybere číslo *k* a vypočítá:

$g^k \bmod q$; $P \otimes \left(g^{y_B}\right)^k \bmod q$
obě čísla zašle *B*

Dešifrování

uživatel *B* vypočítá

$\left(g^k\right)^{y_B} \bmod q$

a určí inverzní prvek. S jeho použitím z druhého čísla zpětně získá *P*.

Korektnost dešifrování

Zřejmě

$$P\left(g^{y_B}\right)^k\left(\left(g^k\right)^{y_B}\right)^{-1} = P\left(g^{y_B}\right)^k\left(\left(g^{y_B}\right)^k\right)^{-1} = P$$

Analýza El Gamalova kryptosystému

kryptosystém je považován za bezpečný,
nevýhodou je nutnost generování náhodných čísel *k* a zdvojnásobení objemu dat při šifrování, je relativně pomalý

el podpis symetricky

Nechť odesílatel *S* zasílá příjemci *R* zprávu *M*.

1. *S* zašle arbitrovi *A* zprávu $E(M, K_S)$
2. Arbiter verifikuje odesílatele a příjemci *R* zašle $E((M, S, E(M, K_S)), K_R)$
3. Příjemce uschová *M* a $E(M, K_S)$ pro účely případného dokazování přijetí

v případě, že nepožadujeme utajení přenášených dat, vystačíme s použitím MAC namísto šifrování

na obranu proti opakovanému použití došlé zprávy lze použít vhodnou časovou známku, proti sestavování nových zpráv z částí dříve došlých poslouží časová známka v každém šifrovaném bloku.

el podepisovana posta

chceme, aby adresát mohl přečíst naši zprávu až poté, co získáme potvrzení o tom, že ji obdržel (elektronický doporučený dopis)

1. *A* zašifruje posílanou zprávu náhodně zvoleným konvenčním klíčem *K*
2. *A* vytvoří 100 párů konvenčních klíčů - první klíč každého páru je generován náhodně, druhý je XOR prvního klíče a klíče *K*
3. *A* zašifruje pomocnou zprávu každým ze 100 párů těchto klíčů (200 šifer)
4. všechny výsledné páry šifer zašle *B*
5. *B* vygeneruje 100 párů náhodných konvenčních klíčů
6. *B* vytvoří 100 párů zpráv tvaru "Toto je levá část mého potvrzení číslo *n*". Opět potvrzení o přjetí je platné, pokud se protější strana může prokázat oběma polovinami jednoho z exemplářů potvrzení.
7. *B* zašifruje *i*-tý pár zpráv *i*-tým párem klíčů ("levou" zprávu jedním, "pravou" druhým z klíčů)
8. výsledné páry šifer zašle *B* protější straně
9. s použitím protokolu pro Oblivious transfer si *A* i *B* navzájem pošlou všech 100 párů svých klíčů - žádná ze stran neví, který klíč z kterého páru protějšek má k dispozici
10. *A* i *B* dešifrují všechny zprávy, ke kterým mají klíče a ověří jejich smysluplnost
11. obě strany si zašlou první bit všech 200 svých šifrovacích klíčů
12. předchozí krok je opakován dokud nejsou přeeseny všechny bity
13. *A* i *B* dešifrují zbývající části párů zpráv, které v předchozích krocích obdrželi - *A* má k dispozici potvrzení o přijetí zprávy od *B* a *B* může provést XOR libovolného páru klíčů a dešifrovat zprávu.

Použití pomocné zprávy dává straně *B* možnost odhalení podvodu v kroku 10.
V ostatních ohledech má protokol obdobné vlastnosti jako protokol pro podepisování kontraktů.

Fish

Systém Fish

proudová šifra založená na Fibonacciho generátoru pseudonáhodných čísel

Vypouštějící generátory (shrinking generators)

předp. dva generátory pseudonáhodných čísel A a S.

A generuje posloupnost $a_0, a_1, \ldots$ prvků $GF(2)^{n_A}$,

S obdobně posloupnost $s_0, s_1, \ldots$ prvků $GF(2)^{n_S}$

dále budeme používat funkci **d**: $GF(2)^{n_S} \rightarrow GF(2)$

Vypouštějící procedura ponechá k dalšímu zpracování prvky a_i a s_i pokud

$$\mathbf{d}(s_i) = 1$$

aplikací vypouštěcí procedury získáme posloupnosti $z_0, z_1, \ldots$ z $a_0, a_1, \ldots$, resp. $h_0, h_1, \ldots$ z $s_0, s_1, \ldots$

Popis algoritmu *Fish*

zvolíme n_A, n_S rovno 32
jako A i S budeme používat uzavřený Fibonacciho generátor

$$a_i = a_{i-55} + a_{i-24} \quad \mathbf{mod}\, 2^{32}$$

$$s_i = s_{i-52} + s_{i-19} \quad \mathbf{mod}\, 2^{32}$$

Samozřejmě prvky $a_{-55}, a_{-54}, \ldots, a_{-1}$ musí být vhodným způsobem odvozeny z klíče. Obdobně pro posloupnost s_i.
Funkce **d** mapuje 32-bitový vektor na jeho nejméně význačný bit.

Není bezpečné používat k šifrování přímo posloupnost $z_0, z_1, \ldots$:
s pravděpodobností 1/8 totiž trojice a_i, a_{i-55}, a_{i-24} projde celá do posloupnosti $\{z_i\}$
rozdělíme posloupnost $z_0, z_1, \ldots$ na páry (z_{2i}, z_{2i+1}), $h_0, h_1, \ldots$ na páry (h_{2i}, h_{2i+1}) a vypočítáme výsledné hodnoty r_{2i}, r_{2i+1}:

$$c_{2i} = z_{2i} \oplus (h_{2i} \wedge h_{2i+1})$$

$$d_{2i} = h_{2i+1} \wedge (c_{2i} \oplus z_{2i+1})$$

$$r_{2i} = c_{2i} \oplus d_{2i}$$

$$r_{2i+1} = z_{2i+1} \oplus d_{2i}$$

Jako tradičně $\oplus$ značí **xor**, $\wedge$ označuje bitové **and**.
Protože h_{2i}, h_{2i+1} mají nejnižší bit jedničkový, je vhodné nastavovat nejnižší bit z_{2i}, z_{2i+1}v závislosti na hodnotě r_{2i}, r_{2i+1}.

Šifrování se provádí např. xorováním výsledné posloupnosti s otevřeným textem.

Analýza

Algoritmus byl publikován koncem roku 1993, nebyl nikdy šířeji používán, není známo, že by existoval efektivní útok.

hasovani

často stačí pouze autentizace původu či obsahu zprávy, případně ověření integrity

ne vždy stačí k zajištění těchto požadavků vlastní šifrovací algoritmus

Použití

- symetrické systémy - slouží k rozpoznání pravosti dešifrované zprávy
- asymetrické systémy - rychlejší autentizace: spočítá se hašovací funkce nad danou zprávou a elektronicky se podepíše až výsledný hashkód
- ochrana hesel a passphrases

Závisí-li výpočet hašovací funkce na tajném klíči, označujeme tuto funkci jako MAC (message authentization code). Pokud takový klíč použit není, jde o MDC (manipulation detection code).

Formální definice

MAC je funkce h splňující

1. zatímco argument X může být libovolné délky, výsledek $h(K, X)$ má pevnou velikost n ($n \geq 32 \ldots 64$)
2. pro dané h a X je těžké určit $h(K, X)$ s pravděpodobností úspěchu výrazněji převyšující $1/2^n$.
3. i se znalostí velkého množství zvolených párů $\{X_i, h(K, X_i)\}$ musí být těžké určit K, nebo spočítat $h(K, X)$ pro $X \neq X_i$.

MDC často dělíme na dvě skupiny - OWHF, CRHF

Funkci h označíme za OWHF (one-way hash function) právě když:

1. zatímco argument X může být libovolné délky, výsledek $h(X)$ má pevnou velikost n ($n \geq 128$, dnes spíše 160 a více)
2. pro dané Y a h musí být těžké nalézt X aby $Y = h(X)$ (tzv. *preimage resistance*)
3. pro dané X a $h(X)$ musí být těžké najít X' aby $h(X) = h(X')$ (tzv. *second preimage resistance*)

hodte korunou

viz oblivious transfer

HW generatory

Generátory založené na speciálním hardware

odvíjejí svoji činnost od sledování nějakého fyzikálního jevu, který je ve své podstatě nepredikovatelný, zhusta dosti sofistikovaně „nevhodným“ způsobem:

- měření záření kousku radioaktivního materiálu
- termální šum polovodiče
- měření stavu volně běžícího oscilátoru
- vnitřní šum zesilovače
- samovolné vybíjení kondenzátoru
- ...

idea

publikován v roce 1991 pod názvem IPES, autoři X. Lai a J. Massey
současný název akronymem za International Data Encryption Algorithm
bloková šifra s délkou bloku 64 bitů, pracující s klíčem o délce 128 bitů
algoritmus je patentován, nelze volně používat

Blok otevřeného textu je rozdělen na čtyři části, každá o délce šestnáct bitů. Poté je provedeno osm kol šifrovacího procesu.

Tvorba subklíčů

celkem 52 subklíčů:
1. klíč rozdělen na osm částí - vznikne prvních osm subklíčů
2. je provedena rotace klíče vlevo o 25 bitů
3. vzniklý řetězec je opět rozdělen na osm částí – subklíčů
4. Opakováním 2 a 3 získáme potřebné množství subklíčů.

Dešifrování

stejný algoritmus jako pro šifrování, rozdíl pouze v použitých subklíčích
použijeme-li pro šifrování v i-tém ($i = 1, \ldots, 8$) kole klíčů

$$Z_1^{(i)}, Z_2^{(i)}, Z_3^{(i)}, Z_4^{(i)}, Z_5^{(i)}, Z_6^{(i)}$$

a v závěrečné fázi $Z_1^{(9)}, Z_2^{(9)}, Z_3^{(9)}, Z_4^{(9)}$,
pro dešifrování použijeme klíčů ve tvaru

$$Z_1^{(10-i)} - 1, -Z_2^{(10-i)}, -Z_3^{(10-i)}, Z_4^{(10-i)} - 1, Z_5^{(9-i)}, Z_6^{(9-i)}$$

a v závěrečné fázi $Z_1^{(1)} - 1, -Z_2^{(1)}, -Z_3^{(1)}, Z_4^{(1)} - 1$.

IDEA může být používána v libovolném pracovním módu pro blokové šifry, zejména v módech ECB, CBC, OFB, CFB
lze použít trojnásobné šifrování EDE Triple-IDEA se dvěma 128-bitovými klíči, použít 52 nezávislých řetězců

Analýza

Je zajímavé, že pokud bychom algoritmus upravili tím způsobem, že zvětšíme délku všech řetězců, se kterými pracuje na dvojnásobek, dojde ke ztrátě bezpečnosti.
Algoritmus je považován za bezpečný.

k cemu je dobre LFSR

Posuvné registry s lineární zpětnou vazbou

(linear feedback shift registers)
sestávají z posuvného registru a vypouštěcí sekvence (tap sequence), což je polynom stupně max. délky posuvného registru
v každém kroku je obsah registru posunut o bit doprava, výstupem je nejpravější bit, registr se zleva doplní o XOR těch bitů v registru, které odpovídají koeficientům vypouštěcího polynomu
pro generátor s maximální délkou nutno polynom volit primitivní polynom stupně n - ireducibilní polynom dělící $x^{2^n-1}+1$, který nedělí $x^d + 1$ pro lib d dělící $2^n - 1$

klice pro konference

Systém distribuce klíčů pro konference komunikujících entit

schopnost vytvořit společný sdílený klíč pro libovolnou skupinu komunikujících - konferenci.

Protokol

umožňuje ustanovit sdílený tajný klíč bez předchozí komunikace jednotlivých členů zamýšlené konference, autentizaci informací, předávaných v rámci procesu ustanovování klíče. Opět je nezbytná existence důvěryhodného centra.

Krok 1: Centrum vygeneruje tři velká prvočísla p, q, r a číslo $n = pq$. Dále nalezne dvojici (e, d) způsobem podobným, jako v kryptosystému RSA tak, aby platilo

$$ed \equiv 1 \pmod{L},$$

$$L = \mathrm{NSD}\big((p-1),(q-1),(r-1)\big), \quad 3 \le e,d < L$$

Rovněž určí prvočíslo c (3 <= c < L) a číslo g, jež je primitivním prvkem v Galoisových okruzích $GF(p)$. $GF(q)$ a $GF(r)$. Pro každou entitu i mající identifikaci I_i centrum spočítá

$$s_i = I_i^d \bmod nr$$

Na závěr tohoto kroku entita i obdrží šestici (n, r, g, e, c, S_i).
Pokud již neočekáváme přijetí žádných dalších entit, je možné čísla p, q a d zrušit. Číslo S_i si každá entita podrží v tajnosti, zbylá čísla jsou společná pro všechny.

Krok 2: V rámci tohoto kroku probíhá vlastní tvorba sdíleného klíče konference.

Analýza
Útok na jednotlivé části protokolu by měl být ekvivalentní s vyřešením alespoň jednoho ze dvou známých obtížných problémů - výpočet prvočíselného rozkladu, nebo výpočet logaritmu nad Galoisovým polem.

kongruencni generatory

Lineární $$X_i = \big(aX_{i-1} + b\big) \bmod m$$
při vhodné volbě *a, b, m* (např. *m* prvočíslo) generuje neopakující se posloupnost délky *m* - *generátor s maximální délkou*

kvadratický $$X_i = \big(aX_{i-1}^2 + bX_{i-1} + c\big) \bmod m$$

kubický $$X_i = \big(aX_{i-1}^3 + bX_{i-1}^2 + cX_{i-1} + d\big) \bmod m$$

kongruenční generátory jsou velmi rychlé, avšak predikovatelné, byla vypracována analýza

konstrukce hasovacich funkci

Hašovací funkce založené na blokových šifrách

výhodné z hlediska návrhu a implementace
většina schémat ekvivalentní s jedním z následujících

Matyas, Meyer, Oseas

$$\mathbf{f} = E^{\oplus}\big(s\big(H_{i-1}\big), X_i\big)$$

délka hashkódu odpovídá délce vstupu, $E^{\oplus}(K, X) = E(K, X) \oplus X$.
ISO/IEC 10118 p. 2

Preneel, Govaerts, Vandenwalle

$$\mathbf{f} = E^{\oplus}\big(s\big(H_{i-1}\big), X_i\big) \oplus H_{i-1}$$

v obou případech E je libovolná bloková šifra, s je funkce mapující prostor zašifrovanách bloků do prostoru klíčů. Tyto funkce mohou být CRHF pokud délka bloku je alespoň 128 bitů. Bezpečnost úměrná hodnotě **min**(k,r).

Hašovací funkce založené na modulární aritmetice

nejlepší schémata založena na umocňování na druhou modulo n

$$\mathbf{f} = \left(X_i \oplus H_{i-1} \right)^2 \mathbf{mod}\, N \oplus X_i$$

použitím většího množství operací lze ještě zvýšit bezpečnost

$$\mathbf{f} = \left(H_{i-1} \oplus \left(X_i \right)^2 \right)^2 \mathbf{mod}\, N$$

Hašovací funkce založené na problému batohu

zatím není jasné, zda problém batohu je těžký pouze v nejhorším případě, nebo zda je těžký v průměrném případě
prakticky všechny systémy založené na problému batohu byly prolomeny

merkle-helmann kryptosystem

založen na problému batohu, přenášená zpráva je chápána jako vektor řešení, přenášena je výsledná suma - "hmotnost batohu"

$a_1, a_2, \ldots a_n$ - posloupnost celých čísel, T cílová suma = hmotnost batohu, hledáme vektor $\boldsymbol{v}$ takový, aby

$$\sum_i a_i v_i = T$$

nechť posloupnost $a_1, a_2, \ldots a_n$ je superrostoucí, problém nalezení vektoru $\boldsymbol{v}$ je v tomto případě zvládnutelný v lineárním čase

Konstrukce systému

zvolíme superrostoucí posloupnost $s_1, s_2, \ldots s_n$, dále vybereme číslo w a modul m,
w bereme nesoudělné s m, $m > s_n$.
Ze zvolených hodnot sestavíme již obecnou posloupnost

$$h_i = w * s_i \mathbf{mod}\, m$$

Posloupnost $\{s_i\}_{i=1\ldots n}$ a čísla w a m utajíme, dále budou sloužit jako soukromý klíč. Posloupnost $\{h_i\}_{i=1\ldots n}$ zveřejníme jakožto veřejný klíč.

Šifrování

Otevřený text P rozdělíme na bloky délky n bitů.
Každý blok P_j nahradíme sumou

$$Cj = \sum_i p_{j_i} h_i$$

Zašifrovaný text C odešleme

Dešifrování

Autorizovaný příjemce vypočítá w^{-1} - z vlastností w a m určitě existuje.
Pro každý blok C_j spočítá $C_j * w^{-1}$.
Vyřeší problém batohu se superrostoucí posloupností $\{s_i\}_{i=1\ldots n}$ pro všechny hodnoty získané v předchozím bodě.
Konkatenací řešení vznikne původní zpráva P.

Poznámky k implementaci

Pro rozumné aplikace: m bývá voleno ve velikosti 100 až 200 číslic,
s_i mají délku 200 až 400 číslic,
batoh mívá přibližně 200 položek

podepisovana posta

Neurčitý přenos (Oblivious transfer)

protokol umožňuje, aby si adresát vybral z několika nabízených možností aniž by odesílatel předem znal jeho volbu, možné doplnění o následnou vzájemnou kontrolu
A vygeneruje dva páry public-key klíčů, oba veřejné klíče zašle B
B vytvoří klíč K pro symetrický algoritmus, tento klíč zašifruje jedním z přijatých klíčů a výsledek vrátí A
A dešifruje přijatou zprávu oběma tajnými klíči, čímž získá K_1 a K_2
Klíčem K_1 zašifruje A jednu z posílaných zpráv, klíčem K_2 druhou a oba výsledky zašle B.
B se pokusí dešifrovat přijaté zprávy, přičemž v jednom případě získá smysluplný výsledek
Případné ověření se provede tak, že A zveřejní své tajné klíče.

Protokol sám o sobě lze používat k distribuci šifrovacích klíčů, případně jako obdobu házení korunou. Větší význam má jako součást následujícího protokolu.

Podepisování kontraktů (Contract signing)

V každém okamžiku musí být obě smluvní strany vázány stejně moc

nejjednodušším řešením je arbitrovaný protokol, kde obě strany předají centrální autoritě své podepsané kopie a tato třetí strana zajistí výměnu po obdržení obou kopií

daleko lepší je následující distribuovaný protokol:
1. A i B náhodně vygenerují 200 konvenčních klíčů, které rozdělí do dvouprvkových množin
2. A i B vytvoří 100 párů zpráv L_n a R_n, zhruba ve tvaru "Toto je levá část n-tého podpisu smlouvy "…, každá ze zpráv navíc obsahuje polovinu elektronického popisu smlouvy, timestamp atd. Kontrakt považujeme za podepsaný druhou stranou, pokud se můžeme prokázat oběma polovinami některého z podpisů.
3. A i B zašifrují i-tý pár zpráv i-tým párem klíčů ("levou" zprávu jedním, "pravou" druhým z klíčů)
4. obě strany si navzájem zašlou páry zašifrovaných zpráv (200 zpráv každý)
5. použitím protokolu pro Oblivious transfer si A a B navzájem zašlou všechny šifrovací klíče - druhá strana má tedy z každého páru jeden (který ?) klíč
6. A i B provedou dešifrování těch zpráv, ke kterým mají k dispozici odpovídající klíč
7. A zašle B první bit všech 200 konvenčních klíčů, obdobně B
8. předchozí krok opakujeme dokud nejsou přeneseny všechny bity klíčů
9. obě smluvní strany mohou dešifrovat všechny zprávy -> kontrakt je podepsán

Pokud by se některá ze stran pokusila o podvod v kroku 4. nebo 5., bude podvod odhalen v kroku 6. Podvod v kroku 7. bude s velkou pravděpodobností objeven okamžitě. Ukončí-li jeden z účastníků protokol před zasláním všech bitů klíčů, mají oba stejnou šanci dopočítat některý z klíčů a získat elektronický podpis druhého.
Problémem je, má-li někdo z podepisujících výrazně větší výpočetní kapacitu, v protokolu není zlom, ve kterém by se výrazně změnila míra vázanosti účastníků.

prahova schemata

protokol, k jehož úspěšnému provedení musí spojit síly více (např. t) účastníků
t nazýváme prahem (threshold value)
pokud $t - 1$ a méně účastníků nemůže získat ani částečný výsledek, mluvíme o *perfektním schématu*

Jednoduché autentizační schéma (2, 2)

nejprve „normální“ algoritmus:

odesílatel a příjemce sdílejí tajný klíč (*a, b*)
autentizační kód zprávy *m* získáme:

$$c = (am + b) \bmod p$$

příjemce přijme *m'* , spočítá

$$c' = (am' + b) \bmod p$$

a ověří, zda *c* = *c'*

totéž jako prahové schéma (2, 2)

a nahradíme dvěma stíny (shadow) s_1, s_2 tak, že $s_1 = a - s_2 \bmod p$, náhodně zvoleno tak, že $0 \le s_2 < p$,
obdobně *b* nahradíme dvojicí s'_1, s'_2
zatímco příjemce si ponechá dvojici *a*, *b*, odesíletelé dostanou dvojici s_1, s'_1 resp. s_2, s'_2.
na vytvoření autentizačního kódu *c* nyní musí spolupracovat:

$$c_1 = (s_1 m + s'_1) \bmod p \text{ resp. } c_2 = (s_2 m + s'_2) \bmod p$$

příjemce sečte obě hodnoty (mod *p*) a dále je protokol stejný, jako předchozí.

uvedené schéma slouží pouze jako ilustrační případ, není bezpečné při vícenásobném použití

Sun a Shieh (t, n) prahové schéma

vedoucí zveřejní velké prvočíslo *p* a generátor α tělesa *T* mod *p*
každý účastník (shadowholder) H_i náhodně zvolí tajný stín s_i a spočítá svůj veřejný parametr

$$p_i = \alpha^{s_i} \bmod p$$

vedoucí následně zvolí (a uchová v tajnosti) náhodný polynom stupně $\mathbf{f}_0(X)$ stupně $t - 1$ nad *T* takový, že jeho konstantní člen je roven sdílené informaci I_0 a náhodný prvek k_0 tělesa *T*.
Na závěr publikuje parametr
…

Původní Sun a Shieh schéma navíc obsahuje mechanismus pro detekci útočníků, kteří by se pokoušeli podsouvat vadný opravdový stín. Tento mechanismus jsme por jednoduchost vypustili, ostatně stejně za jistých okolností způsobuje snížení prahu schématu.

produkcni sifra aka AES

Kryptosystém Rijndael

produkční bloková šifra s proměnnou délkou bloku 16, 24 nebo 32 bajtů a klíčem o délce 128, 192, 256 bitů
založena na podobném principu jako algoritmus Square
Stavem šifry označujeme obsah pole $a_{i,j}$ o rozměrech 4 x (délka_bloku / 32)
Podobně klíč je pole $k_{i,j}$ o rozměrech 4 x (délka_klíče / 32)

blok plaintextu je do stavu vkopírován v pořadí $a_{0,0}$, $a_{1,0}$, atd. podobně klíč
počet kol je závislý na délce bloku a klíče:

Šifrování

Zde:

ByteSub(State)
S-box je nelineární invertibilní transformace definovaná ve dvou krocích –

- provede se invertování (vůči násobení) nad GF(2^8) , 0 je inverzní sama k sobě
- aplikuje se následující transformace

ShiftRow (State)
provádí rotaci řádků 1, 2 a 3 stavu o pevnou hodnotu závislou na velikosti stavu

MixColumn (State)

realizuje násobení sloupce jako polynomu nad GF(2^8) konstantním polynomem $C(x) = 3x^3+x^2+x+2$ modulo x^4+1

AddRoundKey (State, RoundKey)
provádí míchání (XOR) stavu s příslušným podklíčem

Zajímavé je, že celé kolo šifrovacího procesu lze na 32 bitovém procesoru implementovat jako 4 výběry z tabulky a 4 XOR operace

Analýza

Algoritmus byl podroben rozsáhlé analýze a zvolen jako nový standard AES
V současnosti není známa jakákoliv podstatná slabina
Zvláštností je matematický model prováděných transformací

proudove sifry

zpracovávají otevřený text po jednotlivých bitech

RC4

proudová šifra od R. Rivesta, velmi jednoduchý a rychlý algoritmus

používá stavové pole *S* velikosti 256 bajtů (a ještě jedno pro inicializaci klíče) a dva čítače

Inicializace

pole *S* naplníme čísly 0 – 255 (*S*[0] = 0, *S*[1] = 1, …)
pomocné pole *S2* naplníme klíčem *S*[*i*] = *key*[*i* mod keylen]
zamícháme pole *S*:

```
j = 0;
for (i = 0; i < 256; i++) {
        j += S[i] + S2[i] mod 256;
        S[i] ↔ S[j];
}
```

pole *S2* a proměnné *i*, *j* smažeme

Šifrování

```
i = i + 1 mod 256;
j = j + S[i] mod 256;
S[i] ↔ S[j];
output S[ S[i] + S[j] ] mod 256;
```

Analýza

Dosud považováno za velmi kvalitní šifru, není znám jakýkoliv způsob útoku
Jediným nedostatkem špatné statistické vlastnosti prvních cca 100 bajtů výstupu

RC5

publikoval v roce 1994 R. Rivest, přináší novou myšlenku použití rotací závislých na datech
velmi pružný algoritmus s celou řadou parametrů

- délka šifrovacího klíče (0 až 255 bytů)
- počet kol šifrovacího procesu (opět 0 až 255)
- z hodnot 16, 32, 64, ale i vyšších lze zvolit délku slova, algoritmus zpracovává bloky o délce dvojnásobku slova

⊕ označuje bitové XOR, + značí sčítání modulo délka slova, -- odčítání, *x* <- *y* znamená rotaci řetězce *x* vlevo o *y* bitů a symbol -> rotaci opačným směrem.

Šifrování

Předpokládejme prozatím, že máme k dispozici pole subklíčů S. Nechť se blok otevřeného textu skládá ze dvou částí *A* a *B*. Šifrování probíhá dle následujícího předpisu:
…
Velikost slova je rozumné volit v závislosti na velikosti slova používaného procesoru, 128 bitů pro hašování. Šest kol pro nenáročné aplikace (není bezpečné), 32 pro ty nejnáročnější. Jako rozumná se jeví volba délka slova 32 bitů, 12 kol, 16bajtový klíč, což krátce zapíšeme RC5-32/12/16.

Poučení z devadesátých let

- nezbytné zvýšit odolnost vůči diferenční a lineární kryptoanalýze (maskování klíčem, zvýšení počtu kol)
- nově je třeba reagovat na neteoretické fyzikální útoky na procesor realizující šifru (výběr operací ekvivalentně zatěžujících procesor)
- zvýšení efektivity operací využitím plné délky slova procesoru v jednotlivých operacích

RSA

uveřejněný v roce 1977.někdy označován jako kód Herkules
Kryptoschéma je založeno na Eulerově formuli

$$a^{\varphi(n)} \equiv 1 \left(\mathbf{mod}\, n\right) \quad (1)$$

kde $\phi(n)$ je počet čísel z intervalu 1,…, *n*, která jsou s *n* nesoudělná.
Platí:

$$\varphi(\mathrm{n}) = (\mathrm{p}_1 - 1)\mathrm{p}_1^{\mathrm{a}_1 - 1} \cdot (\mathrm{p}_2 - 1)\mathrm{p}_2^{\mathrm{a}_2 - 1} \ldots (\mathrm{p}_\mathrm{k} - 1)\mathrm{p}_\mathrm{k}^{\mathrm{a}_\mathrm{k} - 1} \quad (2)$$

kde

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k} \quad (3)$$

je prvočíselný rozklad čísla *n*.

Šifrování

je třeba znát číslo *n* a malé prvočíslo *e*.
Otevřený text *P* převedeme na posloupnost čísel modulo *n*.
Každý blok P_j zašifrujeme dle vzorce

$$C_j \equiv P_j^{\,e} \quad \left(\mathbf{mod}\, n\right) \quad (4)$$

Spojením výsledných bloků C_j vznikne zašifrovaný text.

Dešifrování

je třeba znát číslo *n*, a číslo *d*.
Každý z bloků C_j dešifrujeme takto:

$$P_j \equiv C_j^{\,d} \pmod{n} \qquad \textbf{(5)}$$

Výpočet dešifrovacího klíče d

Musí platit

$$ed \equiv 1 \pmod{\varphi(n)} \qquad \textbf{(6)}$$

Prvočíslo *e* nesmí dělit ϕ (*n*). *d* určíme z předchozího vztahu rozšířeným Euclidovým algoritmem.
Mimochodem, uvažte následující postup:
Nalezneme

$$r \equiv -\varphi(n)^{e-2} \pmod{e} \qquad \textbf{(7)}$$

Ze (2) plyne $\varphi(e) = e - 1$, s použitím (1)

$$r\varphi(n) \equiv -\varphi(n)^{\varphi(e)} \equiv -1 \pmod{e} \qquad \textbf{(8)}$$

Položíme tedy

$$d = \frac{r\varphi(n)+1}{e} \qquad \textbf{(9)}$$

… tedy existuje více než jeden dešifrovací klíč
V praxi volíme *e* pevné (65535), pro každého účastníka nalezneme zvláštní *n* a dopočítáme dešifrovací klíč. *d* se počítá rozšířeným Euklidovým algoritmem.

SHA-1

message digest funkce transformující zprávu libovolné délky na 160-bitový digest. Pracuje ve třech krocích:

- inicializace – zarovnání vstupu, příprava interních datových struktur
- iterace kompresní funkce – iterativní aplikace kompresní funkce na bloky zprávy, akumulace historie výpočtu
- dokončení – konstrukce finálního výsledku z akumulovaných vnitřních stavů

Inicializace

nejprve se doplní zpráva do násobku 512: doplní se 1, potom tolik nul, aby zpráva byla o 64 krtatší než nejbližší vyšší násobek 512 a na konec do 64 bitů zapsaná původní délka zprávy

Iterace

Kompresní funkce užívá následující interní funkce:
f(X, Y, Z) = (X ∧ Y) ∨ ((¬ X) ∧ Z)
g(X, Y, Z) = (X ∧ Y) ∨ (X ∧ Z) ∨ (Y ∧ Z)
h(X, Y, Z) = (X ⊗ Y ⊗ Z)

Zde ∧ znamená bitové AND, ∨ je bitové OR a ⊗ bitové XOR.

Každý 512-bitový blok zprávy (16 32-bitových bloků plaintextu) je zpracován následujícím algoritmem:

Dokončení

Po zpracování posledního bloku SHA-1 vyprodukuje *digest*:

Na konci roku 2005 byl oznámen útok spočívající v nalezení **kolize** se složitostí 2^{63} operací (Wang, Yao, Yao)
Z praktického hlediska toto neznamená bezprostřední nutnost přestat SHA-1 používat, ale v nových implementacích by měl být upřednostněn některý z novějších algoritmů

SHA 256 (SHA-384, SHA-512)

principiálně stejné algoritmy, jako SHA-1
jednotliví členové rodiny SHAx se od sebe liší délkou slova (32, nebo 64 bitů) a počtem kol 64 u SHA-256, 80 u SHA-512
jde o nově navržené algoritmy, které by měly pokrývat dosud známe slabiny předchozí generace

Příprava vstupních dat

na konec dat připojit bit "1"
doplnit bity "0" do celkové délky ≡ 448 ≡ -64 (mod 512)
doplnit původní délku dat v bitech jako 64-bit big-endian integer

Zpracování zprávy po 512 bitových blocích

každý blok rozložit do šestnácti 32-bitových big-endian wordů $w(i)$, $0 \le i \le 15$

RSA generátor

$$X_i = X_{i-1}^e \bmod n$$

výsledkem nejnižší bit X_i. Bezpečnost ekvivalentní s bezpečností RSA

rysy navrhu krypt. Protokolu

použijeme-li při tvorbě systému k řešení nějakého problému odpovídající protokol, stačí pouze ověřit korektnost implementace vzhledem k tomuto protokolu

1. *arbitrované protokoly* - *arbitrem* rozumíme neutrální třetí stranu, zajišťující férovost, nevýhodou použití arbitra jsou zvýšené náklady na provoz, problémy s dostupností, časové prodlevy, potíže s důvěryhodností a výkonem

2. *rozhodované (adjudicated) protokoly* - rozhodčí (adjudicator) je třetí strana, která je schopna rozhodnout, zda operace byla vykonána férově, případně kdo z účastníků porušuje pravidla, třetí strana je tedy používána pouze při sporu

3. *samozabezpečovací (self-enforcing) protokoly* - samotný protokol zajišťuje vzájemnou ochranu účastníků

statisticke testy generatoru

Monobit test

zda počet „1“ a „0“ je přibližně ‚správný’

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

zde n_0 je počet „0“, n_1 je počet „1“ ve vzorku délky n.

Serial test

zda počet výskytů „00“, „01“, „10“ a „11“ je přibližně stejný, jednotlivé podřetězce se mohou překrývat.

$$X_2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

zde n_{xx} výskytů řetězce „xx“ ; $n_{00} + n_{01} + n_{10} + n_{11} = n - 1$

Poker test

zkoumá, zda posloupnost obsahuje zhruba stejný počet všech možných podposloupností délky m

buď m tž. $\left\lfloor \frac{n}{m} \right\rfloor \geq 5\left(2^m\right)$ a položme $k = \left\lfloor \frac{n}{m} \right\rfloor$

vstupní posloupnost rozdělíme na k nepřekrývajících se podposloupností délky m, označíme n_i počet výskytů i-tého typu podposloupnosti

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

Runs test

zda vstupní posloupnost obsahuje správný počet běhů, tj. posloupností samých nul (díry) a jedniček (bloky);

očekávaný počet běhů délky i je $e_i = \left(n - i + 3\right)/2^{i+2}$

nechť k je nejvyšší hodnota i tž. $e_i \geq 5$.

$$X_4 = \sum_{i=1}^{k} \frac{\left(B_i - e_i\right)^2}{e_i} + \sum_{i=1}^{k} \frac{\left(G_i - e_i\right)^2}{e_i}$$

Zde B_i je počet bloků, B_i počet děr délky i.

Autocorellation test

cílem je srovnání korelace mezi vstupní posloupností s a její posunutou verzí

buď d tž. $1 \leq d \leq \lfloor n/2 \rfloor$, počet bitů, ve kterých se s liší od posunuté verze označíme

$$A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$$

$$X_5 = 2\left(A(d) - \frac{n-d}{2}\right) \Big/ \sqrt{n-d}$$

utok na generátor

Choosen input atttack

Útočník ovládá nebo alespoň všechny, nebo některé zdroje entropie generátoru, pokud má přístup k výstupu, může provádět adaptivní útok

Kryptoanalytický útok

Hledání charakteristiky výstupní posloupnosti, predikovatelnosti způsobené nesprávným návrhem generátoru, odpovídá lámání proudových šifer

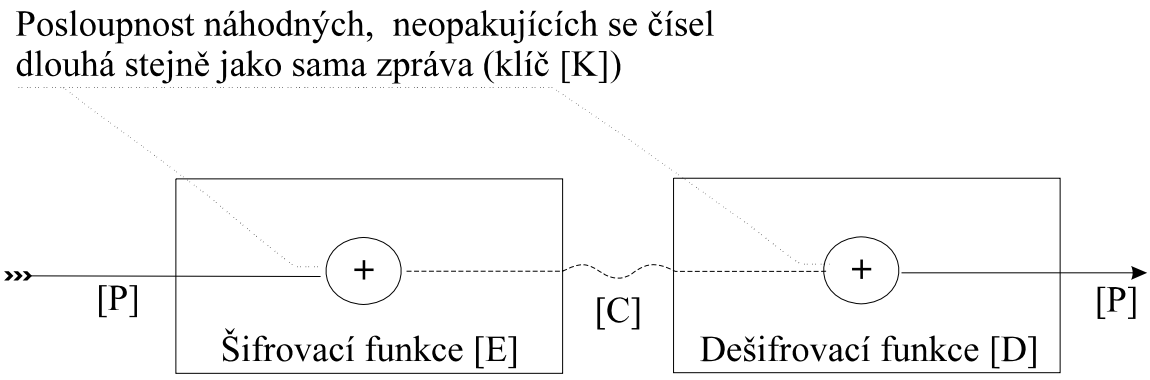
Iterative guessing

Pokud při překlíčování generátor nezahrne dostatečné množství nové entropie, je změna vnitřního stavu generátoru „příliš malá“

Pokud útočník znal stav generátoru před změnou, dokáže nalézt stav po změně a nadále predikovat výstup

vernamova sifra

možná implementace one-time pad-u



Yarrow

Generátor Yarrow

uvádím jako příklad komplexní konstrukce náhodného generátoru
cílem útočníka je získat vnitřní stav generátoru, pokud uspěje, může predikovat výstup generátoru, nebo rekonstruovat dřívější části výstupní posloupnosti
kvalita generátoru závisí na kvalitě vstupu, častým problémem nesprávný odhad míri entropie zdrojů
generátor se skládá ze čtyř hlavních komponent:

- akumulátor entropie (entropy accumulator)
- mechanismus změny klíče (reseed mechanism)
- mechanismus generování (generation mechanism)
- řízení změny klíče (reseed control)

cílem je kvalitní akumulace entropie ze všech zdrojů, zajištění dostatečně častých (nealgoritmických !) změn klíče a občasná změna klíče, která bude pro útočníka nepřekonatelná i v případě, že u některého zdroje došlo k nadhodnocení obsahu entropie



The Yarrow algorithm is a cryptographically secure pseudo-random number generator. The name is taken from the yarrow plant, the stalks of which are dried and used as a randomising agent in I Ching divination.

It was designed by Bruce Schneier, John Kelsey, and Niels Ferguson of Counterpane Labs (Kelsey et. al, 1999). The Yarrow algorithm is explicitly unpatented and royalty-free; no license is required to use it. Yarrow is incorporated in Mac OS X and FreeBSD for their /dev/random devices.

An improved design from Ferguson and Schneier, Fortuna, is described in their book, Practical Cryptography.

Autor neručí za správnost, úplnost a ani za to, že je popisováno opravdu to, co je ve žlutých nadpisech :)