Activiteit waarbij voorzichtigheid geboden is

Database-encryptie

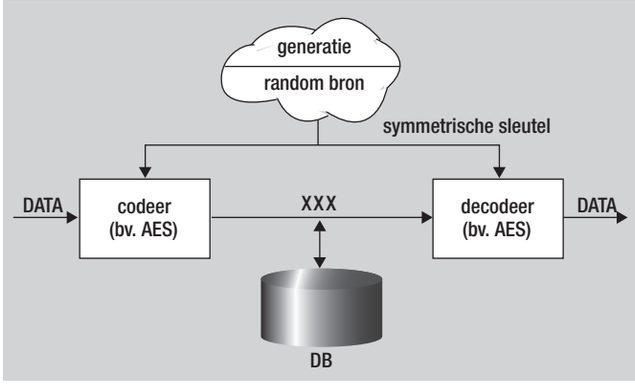
Rick van Rein

Het kan voor de vervulling van randvoorwaarden aan een database-ontwerp zinnig zijn om data te encrypten, bijvoorbeeld ter bescherming van privacy of commerciële belangen. Alleen is de uitwerking daarvan in een database vergeven van voetangels en klemmen, zoals we uit de doeken doen in dit uitstapje naar de cryptografie.

Het toepassen van encryptie op databases houdt in dat de erin opgeslagen data onleesbaar worden gemaakt voor onwelkome bezoekers. Een database biedt toegangsrechten om dit te regelen, maar soms volstaat dat niet, bijvoorbeeld wanneer vereist wordt dat ook directe toegang tot de onderliggende opslagstructuren beschermd zijn of wanneer iemand die een password raadt of weet bepaalde data niet mag inzien. In zo'n geval past men encryptie toe.

Symmetrisch encrypten

Het doel van encryptie is om data te veranderen in ruis, of in elk geval iets wat geen patroon of gegevens meer lijkt te bevatten. Alleen door een decryptiebewerking uit te voeren met de juiste sleutel kunnen de originele data worden teruggehaald. Dit is een voor de hand liggende wens, maar bepaald niet eenvoudig te realiseren in de specifieke context van een database.
De eenvoudigste vorm van encryptie die we kennen is de zogenaamde symmetrische encryptie, die zo genoemd is omdat dezelfde sleutel wordt gebruikt voor encryptie en decryptie. Een stuk data kan met zo'n sleutel door een encryptie-algoritme worden gemangeld, wat iets oplevert van ongeveer dezelfde lengte wat veilig kan worden opgeslagen. Wanneer zulke gecodeerde data vervolgens worden aangeboden aan een decryptie-algoritme die dezelfde sleutel gebruikt, kunnen de originele data worden gereconstrueerd.



Afbeelding 1: Symmetrische encryptie gebruikt dezelfde sleutel voor encryptie en decryptie.

Iedereen zal in zijn jeugd met een vriendje weleens een geheimtaal hebben bedacht, bijna iedereen heeft wel eens gespeeld met het verschuiven van letters over een aantal posities in het alfabet. Het aantal posities is hierin eigenlijk de sleutel die aangeeft hoe de gecodeerde en ongecodeerde informatie samenhangen, gegeven het algoritme van verschuiven-binnen-het-alfabet. In sterkere encryptie-algoritmen is een sleutel een soort 'husselvoorschrift' op de te coderen gegevens. Het geeft aan hoe de bits worden verwisseld en met elkaar gecombineerd. Al die bewerkingen zijn omkeerbaar, zodat een decryptie-algoritme kan bestaan dat de werking van het encryptie-algoritme omkeert.
Symmetrische encryptie vindt doorgaans plaats in vaste datablokken. Tegenwoordig worden we gelukkig van blokken van 128 bits lang, en de sleutel kiezen we liefst ook als een willekeurig getal van minimaal die lengte. Die afmetingen hebben te maken met wat momenteel en in de aanstormende toekomst kraakbaar is met beschikbare computerkracht.
Behalve het aantal bits op diverse plaatsen is ook het algoritme bepalend voor de kraakbaarheid; van WEP (in gebruik bij Wireless LAN's) is bekend dat het in een paar uur kraakbaar is op een gewone PC, want het is een uitermate zwak algoritme. DES heeft het tot ongeveer 1982 uitgehouden, maar geldt nu ook als onveilig, voornamelijk doordat de sleutels te kort zijn. Er is nog een patch toegepast met 3DES ('triple DES'), maar tegenwoordig werkt men met AES, een gedegen encryptiestandaard die andere algoritmen zal opvolgen en vervangen. AES staat voor Advanced Encryption Standard en is onder vuur genomen door vooraanstaande cryptografen, en daarbij niet 'te licht' bevonden.

Dagelijks gebruik van symmetrische sleutels

Wie data in encrypted vorm wil opslaan zal de sleutel moeten hebben waarmee de encryptieslag gedaan kan worden. Als het gaat om symmetrische encryptie is dat ook meteen de sleutel waarmee decryptie plaatsvindt. Dat houdt in dat er geen

scheiding te maken is tussen schrijvers en lezers van data; iedereen heeft dus dezelfde rechten over gedeelde data.
Het is denkbaar om meerdere sleutels te gebruiken zodat gescheiden segmenten in de database ontstaan, waarin groepen gebruikers toegang kunnen krijgen tot een segment zonder noodzakelijk bij andere segmenten te kunnen. Wanneer van data bekend is tot welk segment het behoort (bijvoorbeeld doordat alles in een bepaald attribuut in segment A hoort, of doordat een identifier van een segment bij de data genoemd staat) dan kunnen de gebruikers die toegang hebben de desbetreffende sleutel opzoeken en gebruiken om de data te decrypten.
De sleutels die per segment worden gebruikt zijn willekeurig geprikte getallen, die dezelfde zorgvuldige afhandeling krijgen van (gedeelde) passwords. Want eigenlijk werken ze precies zo – het zijn altijd dezelfde geheimen, en altijd worden ze letterlijk gebruikt in de encryptie- en decryptie-algoritmen.

Een probleem dat specifiek geldt voor de toepassing binnen databases, is dat de data die beschermd worden vaak fragmentarisch zijn opgeslagen. Hele kleine stukjes worden in afzonderlijke attributen opgeslagen, die elk afzonderlijk kunnen worden opgevraagd en aangepast.
Een probleem bij gefragmenteerde opslag is dat de dichtheid van de informatie per attribuut vaak te wensen overlaat (zie het kaderstuk) zodat het regelmatig nodig zal zijn om extra bits toe te voegen om de volle kracht van het encryptie-algoritme te benutten. Dit vergroot de hoeveelheid opgeslagen data, en dat kan drastische invloed hebben op de efficiënte werking. Een extreem voorbeeld is een attribuut dat 'true' of 'false' kan zijn maar dat moet worden aangevuld tot 128 bits om een volledig blok aan AES aan te bieden.

> Encryptie-algoritmen munten uit in hun niet-lineaire rekenintensiteit wanneer de sleutel onbekend is

Een ander obstakel is dat een database vaak diverse attributen met dezelfde data bevat, en dat het als een lek kan worden beschouwd wanneer deze links ook afleidbaar zijn door iemand zonder de decryptie-sleutel. Ook dit probleem is een direct gevolg van de uit het relationele model voortvloeiende wens om attributen afzonderlijk te encrypten. Cryptografen werken daarom het liefst met een volledig datablok met willekeurige data dat voorafgaat aan de te coderen data; wederom een potentieel grote opoffering aan opslagruimte op schijven en in caches.

Zoeken

Het grootste probleem zit hem echter niet in het ruimtebeslag maar in de mogelijkheid om in de data te zoeken. Afhankelijk van de manier waarop encryptie is geïmplementeerd zal een database ofwel geen enkele sleutel kennen en dus even weinig verbanden zien als een buitenstaander, ofwel hij zal alle gebruikte sleutels kennen en ze voortdurend moeten toepassen op de data die ingelezen worden. Dat laatste zou geen ramp zijn wanneer het alleen de voor een query op te lepelen data betrof, maar met SQL wil men doorgaans tabellen koppelen op basis van attributen, en daarvoor moet door de te koppelen attributen worden geïtereerd. Als alle data die meespeelen in zo'n iteratie niet in de geheugencache passen, dan moet dezelfde pagina meermalen ingelezen worden van de schijf, en bij gebruik van encrypted data houdt dat in dat de decodering dan ook meermalen moet worden gedraaid.
De mechanismen van encryptie op databases dienen vooral om de kraker de mogelijkheid te ontnemen om met brute kracht achter encrypted data te komen, maar datzelfde zit de database ook in de weg bij het uitvoeren van massale query's. Dat is een fundamenteel probleem – brute kracht-aanvallen werken door het decrypten van data met allerlei mogelijke sleutels, en dat lijkt wel heel sterk op een massale query. Toch is het niet hopeloos, want ook de meest massale query's zijn nog ordes van grootte kleiner dan een kraakpoging, en de eigenschap waarin encryptie-algoritmen uitmunten is hun niet-lineaire rekenintensiteit wanneer de sleutel onbekend is.
Het is verder maar in beperkte mate mogelijk om een index te maken over encrypted data. Wie wil weten of zijn geheime

De dichtheid van informatie

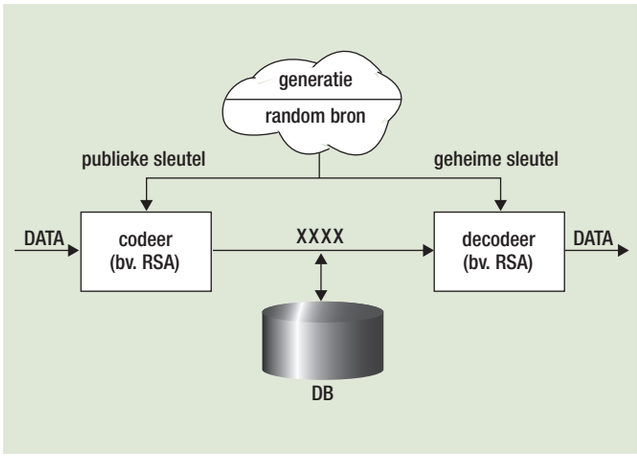
Een aanname onder alle encryptievormen is dat de gecodeerde informatie maximale dichtheid heeft en desondanks een minimumgrootte haalt. Dit voorkomt dat alsnog patronen ontstaan in de gecodeerde informatie, wat zou leiden tot te zwakke encryptie. Een voorbeeld kan dat verduidelijken.
Stel, een bank wil kunnen garanderen dat ze geen PIN-codes opslaat in haar databases. Dat kan ze verwezenlijken door alleen maar encrypted PIN-codes op te slaan. Nu zit er in een PIN maar vrij weinig informatie; er zijn maar 10^4 codes mogelijk in 4 cijfers, veel minder dus dan de 2^{128} van een datablok in een modern encryptie-algoritme. Wanneer de PIN naïef wordt gecodeerd, ontstaan dus wel gecodeerde blokken van 128 bits, maar er zijn nog steeds maar 10^4 mogelijke blokken. Een kraker zou er voor kiezen om een opzoektabel met 10^4 vermeldingen te construeren en daarmee de gecodeerde informatie te herleiden tot de originele PIN.
Wat een bank daarom doet, is op elke PIN-pas een random bitreeks toevoegen die het aantal informatie-bits tot tenminste de blokgrootte van het encryptie-algoritme uitbreidt. Dat is gewoon een willekeurig getal dat ooit bedacht is, en dat na koppeling aan de PIN gecodeerd kan worden tot hetgeen in de database van de bank gecontroleerd kan worden.

attribuut voorkomt in de database kan nog wel zoeken naar de encrypted vorm daarvan, maar andere selectie-criteria dan gelijkheid en ongelijkheid zijn uit den boze; ten gevolge van het husselen van bits is volgorde bijvoorbeeld niet meer terug te vinden in de encrypted data. Zo hoort het natuurlijk ook vanuit de invalshoek van de encryptie, maar het beperkt de mogelijkheden van indexen enorm.

Tenslotte klinkt het opsplitsen van data in segmenten op zich aardig, maar het kan problemen geven als er naast gebruikers ook half-vertrouwde processen tussen zitten. Denk bijvoorbeeld aan webservers die opstarten zonder menselijke tussenkomst, en dus ook zonder een buiten het systeem gehouden password. Zulke processen bieden een pad waarlangs de computer (en dus ook een kraker) toegang kan krijgen tot alle data in een segment. Dit kan wel eens strijdig zijn met de originele doelstelling achter de encryptie van dat segment.

Public-key crypto

Symmetrische encryptie biedt sterke beveiliging van data, maar laat het soms afweten omdat toegang tot datasegmenten voor lezers en schrijvers van data gelijk is. Wie dus creditcard-nummers kan wegschrijven, kan ze ook teruglezen. Om dat op te lossen bestaat een reken-intensievere oplossing in de vorm van public-key crypto.



Afbeelding 2: Public-key crypto gebruikt afzonderlijke sleutels voor encryptie en decryptie.

Dit mechanisme gebruikt een sleutelpaar in plaats van een enkele sleutel. De ene sleutel uit het paar wordt openlijk gepubliceerd en dient voor encryptie, terwijl de andere als geheime sleutel geldt waarmee de encryptie kan worden teruggedraaid. De twee sleutels zijn dus gerelateerd, maar na de generatie van het sleutelpaar kunnen de verbandleggende gegevens worden vernietigd omdat de sleutels ook zonder dat als paar gebruikt kunnen worden.
Het grote goed van deze technologie is dat 'zij-die-kunnen-encrypten' losgekoppeld worden van 'zij-die-kunnen-decrypten'. Daarmee is een veel flexibeler encryptie-model mogelijk dan met symmetrische encryptie, zie het kader over opslag van creditcard-nummers voor een voorbeeldtoepassing die dit benut.

De sleutels die per segment worden gebruikt zijn willekeurig geprikte getallen

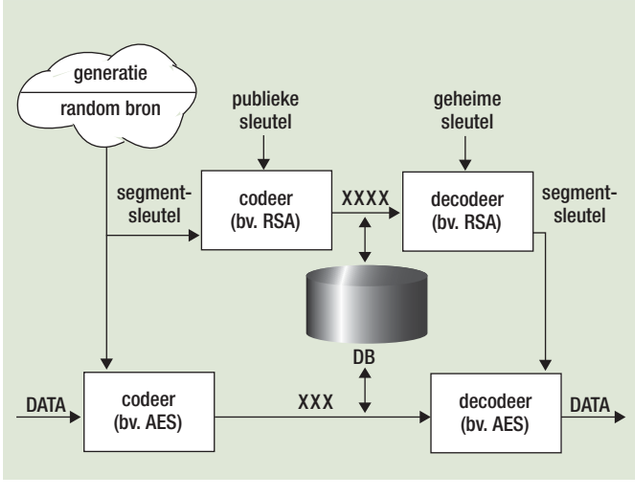
De genoemde nadelen van symmetrische crypto zijn ook van toepassing op public-key crypto, met als extra probleem dat de berekeningen veel zwaarder zijn. Een AES-blokbewerking kan in een microseconde worden gedaan door gespecialiseerde hardware, terwijl het meestgebruikte public-key algoritme RSA met gemak millisecondes verslindt, tot een seconde in nadelige gevallen.
Alsof dat nog niet genoeg problemen geeft zijn de datablokken voor RSA ook groter (minimaal 1024 bits) en voor de afzonderlijke verwerking van losse attributen vormt dat dus nogal wat overhead.
Public-key crypto is daarom een techniek die alleen moet worden gebruikt als het echt niet anders kan; het is krachtig maar kostbaar. Een technische uitweg daarvoor is een combinatie van symmetrische en public-key technieken, welke het meest optimaal is voor een bepaalde toepassing.

Wat meestal wordt gedaan is een willekeurige symmetrische sleutel prikken voor een segment, waarna die sleutel encrypted wordt met een publieke sleutel. Hierna is de zojuist geprikte symmetrische sleutel te gebruiken om alle data in een segment te coderen. Dit creëert dynamisch nieuwe segmenten, die alleen toegankelijk zijn voor de houders van de geheime sleutel die gepaard is met de gebruikte publieke sleutel.

Wie, wat, waar?

In het voorgaande zijn de mechanismen besproken om data te encrypten, plus de gevolgen daarvan voor toepassing in databases. Een punt dat nog onbesproken is gebleven, is de plek waar deze algoritmen ingebouwd kunnen worden. Dit is vooral een vraag naar de plek waar het vertrouwen in het correct handelen met sleutels wordt gelegd.
Zoals aangegeven kan een database weliswaar met kennis van sleutels handelen, maar levert dat nauwelijks efficiëntere afhandelingsmogelijkheden op, doordat indexen toch alleen op gelijkheid kunnen worden doorzocht. Daarvoor volstaat encryptie buiten de database evengoed als encryptie er binnen. Een argument dat het toch aantrekkelijk kan maken om de encryptie binnen een database te laten plaatsvinden, is dat een ondersteunende database het mogelijk transparant maakt. Dat scheelt dus complicaties in de SQL-code die de database aanspreekt, en het lokaliseert de encryptie-zorgen in de database. De zorgen over het afbakenen van segmenten en het kiezen van een encryptie-techniek kan dit echter niet wegnemen, want die problemen zijn universeel.

Er valt ook wel wat voor te zeggen om de kennis over decryptie buiten de database te houden, met name als het gaat om public-key crypto. Wanneer decryptie namelijk buiten de database blijft, is het niet meer noodzakelijk om de database en het systeem waarop het draait te vertrouwen om goed met sleutels en gedecodeerde data om te springen. Dat levert niets op voor bedrijfsinterne databases, maar des te meer voor databases die extern toegankelijk zijn, bijvoorbeeld via het Internet.
Wanneer cryptografische bewerkingen naar de gebruikers toe worden getrokken ontstaat zelfs de mogelijkheid om heel verfijnde sleutelbeheerstechnieken toe te passen, zoals over personen gesplitste sleutels of de generatie en opslag van geheime sleutels op een als USB-device uitgevoerd token. Hoewel dat te ver voert om hier te bespreken, komt het er op neer dat de variatiemogelijkheden verbeteren als de cryptografie afzonderlijk van een voorgestanste database wordt uitgewerkt.



Afbeelding 3: Symmetrische encryptie met een sleutel die via public-key crypto afgeschermd is.

Casus: creditcard-nummers opslaan

Een belangrijke toepassing van encryptie van database-inhoud is de encryptie van creditcard-nummers in webservers. Deze nummers vormen als 'license to withdraw' immers een belangrijke attractie voor krakers.
Aangezien deze gegevens per stuk worden toegevoegd, en doorgaans van een online web-winkel *downloaded* worden voor verwerking elders, is de web-winkel alleen een tijdelijke opslag voor deze gegevens. Het is dus niet belangrijk voor de web-winkel om te kunnen zoeken in de opgeslagen creditcard-nummers of om ze te kunnen presenteren op de web-interface.
De oplossing in dit geval is daarom vrij eenvoudig – encrypt de creditcard-gegevens met een publieke sleutel en plaats de uitkomst van die operatie in de database. Mits gedaan volgens de regels der kunst kan geen kraker daar nog wijs uit worden. Na download naar de verwerkingslocatie kunnen de gegevens weer gereconstrueerd worden, met een daar aanwezige geheime sleutel die hoort bij de publieke sleutel die bij de encryptie werd gebruikt.

Conclusies

Er is veel mogelijk op het gebied van encryptie van gegevens in een database. Maar het is een activiteit waarbij voorzichtigheid geboden is; allereerst vanwege de niet-geringe kans om schijnveiligheid te creëren en daarmee krakers alsnog toegang tot data te verschaffen, verder vanwege de invloed die het heeft op de efficiënte werking van de database.
Het is daarom beter om de toepassing van encryptie in een database tot een minimum te beperken. Verder dient al in een vroeg stadium een analyse gemaakt te worden van de toegangswensen voor de encrypted data. Gegeven een degelijke voorbereiding kan encryptie dan wel degelijk waarde toevoegen aan een databaseontwerp.

Rick van Rein
Dr. ir. H. van Rein (rick@openfortress.nl) is ontwikkelaar en beheerder bij OpenFortress Digital signatures.