Cosa fare quando un algoritmo crittografico viene 'rotto' ?

Nell'agosto 2004 alcuni ricercatori cinesi hanno annunciato di aver scoperto dei problemi negli algoritmi di Hash MD4, MD5, HAVAL-128 e RIPEMD.[1] Gli stessi ricercatori hanno annunciato all'inizio di Febbraio 2005 di aver scoperto degli altri problemi nell'algoritmo di Hash SHA-1. Nella rubrica Crittografia Moderna in questo numero i lettori potranno trovare una discussione dei problemi di SHA-1 e del loro significato più tecnico. In questa rubrica vorremmo invece considerare più direttamente gli aspetti pratici di questi risultati.

Cosa vuol dire '*rotto*' ?

Prima di tutto è necessario chiarire il significato del termine '*rotto*'. Per un crittografo o crittoanalista questo termine indica che un algoritmo non soddisfa ai requisiti per i quali era stato concepito. Questo però non vuol dire che in termini pratici sia possibile *rompere* l'algoritmo. Consideriamo ad esempio l'annuncio dei ricercatori cinesi, ripreso da Bruce Schneier, a proposito di SHA-1. In questo annuncio si dice che è possibile trovare delle *collisioni*[2] in SHA-1 utilizzando 2^{69} operazioni di Hash invece delle 2^{80} promesse come minimo dall'algoritmo. E' questo un vero problema pratico oggi? Per renderci conto di cosa voglia veramente dire questo numero, confrontiamo questo dato con il medesimo valore per MD5 il quale promette di richiedere non meno di 2^{64} operazioni di Hash. Quindi SHA-1 'rotto' sarebbe ancora più resistente di MD5 'non rotto'! (MD5 comunque soffre di problemi analoghi, ed ora si ritiene che il minimo numero di operazioni di Hash necessario per MD5 sia solo 2^{42}.) In entrambi i casi, oggi non sembra possibile implementare praticamente questi attacchi, anche se in tempi brevissimi, anche solo qualche mese, gli sviluppi teorici da un lato e la crescita di potenza degli elaboratori dall'altro potrebbero rendere alcuni di questi attacchi possibili almeno su super-computer.

1 Si veda la rubrica Crittografia Moderna di ICT Security, Novembre 2004.

2 Si veda la rubrica Crittografia Moderna su questo numero per maggiori dettagli su questo punto.

Dalla *rottura* ai rischi

Il problema principale per applicazioni commerciali comuni, è un altro. Se è stato dimostrato che un algoritmo crittografico non soddisfa i requisiti per i quali era stato concepito, c'è il *rischio* che prima o poi si trovi il modo di romperlo anche praticamente. Nel caso di MD5 o SHA-1, ciò vuol dire che il numero necessario di operazioni per trovare delle collisioni si riduca sufficientemente per renderlo possibile su di un elaboratore qualsiasi. Chi si occupa di sicurezza informatica dovrebbe quindi porsi *immediatamente* il problema di cosa fare quando vengono annunciati risultati di questo tipo. Ovviamente il primo passo è di valutare i rischi che si corrono nel utilizzare un algoritmo crittografico che potrebbe rilevarsi del tutto inadeguato in tempi *brevi*. Ovviamente i rischi dipendono sia dall'uso che si fa dell'algoritmo che dai tempi per i quali si vuole che le sue funzioni persistano. Gli algoritmi di Hash ci offrono lo spunto per approfondire questo argomento.

Gli usi degli algoritmi di Hash

Dato un documento di lunghezza arbitraria, un algoritmo crittografico di Hash produce una stringa di lunghezza fissa, detta anche *impronta*, con le seguenti caratteristiche:

1. è praticamente impossibile ricostruire il documento a partire dall'impronta,
2. l'impronta è praticamente unica per ogni documento,
3. una piccola modifica in un documento genera una grande modifica nell'impronta.

Un algoritmo di Hash deve soddisfare queste proprietà in modo tale che non sia possibile violarle con le potenze di calcolo a disposizione oggi e nel prossimo futuro.

Queste proprietà sono usate per molti scopi. Il principale è garantire la integrità di un documento, file, pacchetto di rete eccetera. Confrontando i dati con la loro impronta è possibile dedurre se i dati sono stati modificati o meno. Ad esempio, i produttori di software forniscono le impronte dei programmi che vendono o distribuiscono. Se l'algoritmo di Hash utilizzato non garantisse le proprietà richieste, un attaccante potrebbe modificare i programmi senza che vi fosse modo di

rilevarlo, soprattutto se questi sono distribuiti solo come codice eseguibile e senza codice sorgente.
Un altro esempio dell'uso degli algoritmi di Hash sono i file di password. Di norma le password non sono memorizzate in chiaro negli elaboratori, ma vengono memorizzate solo le impronte delle password. Ogni volta che un utente si vuole autenticare, il sistema calcola l'impronta della password fornita e la confronta con quella registrata. Supponiamo che un attaccante riesca ad ottenere copia del file di password con tutte le impronte. Se l'algoritmo di Hash utilizzato soddisfasse le proprietà indicate, un attaccante potrebbe solo provare tutte le password e confrontarle con l'impronta,[3] ma in caso contrario potrebbe facilmente risalire direttamente dalle impronte alle password. Le impronte sono utilizzate da tutti i protocolli di comunicazione per garantire che i dati non siano modificati durante la trasmissione. In questo caso non vi sono particolari richieste sui tempi visto che di norma una volta che i dati sono arrivati e l'impronta verificata non vi è necessità di mantenerla nel tempo. I *File Integrity Checker* invece verificano l'integrità dei dati sui dischi o altri supporti utilizzando anch'essi le impronte, in questo caso vi sono situazioni in cui è necessario mantenere l'integrità, e quindi le impronte, per lunghi periodi di tempo.
L'applicazione che forse ha i maggiori problemi temporali è quella delle firme digitali, visto anche il loro utilizzo e valore legale. Una firma digitale è realizzata cifrando con la propria chiave privata l'impronta di un documento. In questo modo chiunque è in possesso della chiave pubblica può decifrare l'impronta e verificare se coincide con quella calcolata a partire dal documento. Se l'algoritmo di Hash non soddisfa le proprietà richieste, potrebbe essere possibile creare un altro documento con la stessa impronta, e quindi anch'esso firmato digitalmente dalla stessa persona, ma ovviamente a sua insaputa! Vogliamo sottolineare che al giorno d'oggi gli attacchi a MD5 e SHA-1 non permettono di fare esattamente questo, in quanto essi permettono di costruire documenti con la stessa impronta che differiscono però per caratteri binari particolari. E' difficile pertanto oggi utilizzare questi attacchi se non in alcune situazioni molto particolari.

3 In ogni caso, il numero di password da provare sarebbe comunque limitato a causa della ben nota difficoltà umana a scegliere password valide; se un attaccante ottiene copia di un file di password è quindi necessario cambiare immediatamente tutte le password.

Cosa si può fare

Poiché la crittografia ha un approccio proattivo alla sicurezza, bisogna avere un approccio proattivo alla crittografia, in altre parole sempre guardare molto avanti. Detto ciò, possiamo prendere alcune precauzioni, a seconda dell'utilizzo che facciamo dell'algoritmo crittografico in questione.

In generale, la prima cosa da fare è avere già previsto percorsi per sostituire l'algoritmo con un altro, magari anche solo cambiando la lunghezza delle chiavi usate per gli algoritmi di cifratura. Molti produttori di software stanno adottando questo approccio, ed in occasioni come queste rilasciano in tempi brevi una nuova versione del loro prodotto che utilizza un diverso algoritmo mantenendo però la compatibilità con i precedenti. Ovviamente, a parte il lavoro del produttore, è necessario che anche l'utilizzatore sia attento ad aggiornare i propri programmi in tempi brevi.

Per chi utilizza invece direttamente gli algoritmi, si devono prendere alcune iniziative immediate e si dovrebbero adottare alcune strategie preventive. La prima cosa da fare è quella di sostituire MD5 e SHA-1. Già l'NSA americana aveva annunciato che entro il 2010 SHA-1 sarebbe stato abbandonato, i nuovi risultati ne accorceranno sicuramente la vita. I sostituti indicati dall'NSA sono SHA-256 e SHA-512 (basati su algoritmi matematici sufficientemente diversi da sperare che non soffrano degli stessi problemi di SHA-1), oppure altri algoritmi sono RIPEMD-160, Whirlpool e Tiger. Il problema con questi sostituti è che essendo algoritmi per lo più nuovi non sono stati studiati in profondità, soprattutto Whirlpool e Tiger che adottano algoritmi molto diversi dai più tradizionali SHA e RIPEMD.

Oltre a sostituire gli algoritmi problematici, è necessario instaurare delle strategie preventive. La principale strategia preventiva è di utilizzare due diversi algoritmi per lo stesso scopo e di unire i loro risultati, ad esempio con un XOR. Ovviamente il programma deve fare il doppio del lavoro, ma per poterlo *rompere* è necessario che entrambi gli algoritmi siano rotti allo stesso tempo. Questo approccio è stato preso ad esempio nella definizione della funzione PseudoRandom nel Protocollo TLSv1.0, utilizzato comunemente dai browser web per cifrare le connessioni ai siti di commercio elettronico, che utilizza sia MD5 che SHA1 in parallelo facendone poi l'XOR dei risultati.

Ovviamente è anche possibile utilizzare algoritmi molto più *forti* di quanto sia necessario oggi nell'ottica che questi possano durare più a lungo nel tempo anche se saranno scoperti dei problemi di sicurezza simili a quelli odierni. Ad esempio, SHA-512 promette di richiedere non meno di 2^{256} operazioni di Hash (un numero enorme) per creare una collisione, ed anche se questo algoritmo avesse lo stesso problema di SHA-1 con una riduzione di 2^{11} operazioni, richiederebbe sempre 2^{245} operazioni.

Come abbiamo visto, i problemi maggiori sono però per le applicazioni di firma digitale. In linea di principio le firme digitali, per il loro valore anche legale, dovrebbero valere per sempre, od almeno per lunghi periodi. In questo caso il problema non è solo tecnico, ma anche e soprattutto normativo. Dal punto di vista tecnico quello che si può fare è generare firme digitali con più di un algoritmo e richiedere che tutti gli algoritmi verifichino la firma, ed introdurre procedure per poter aggiornare le firme nel tempo, ovvero riapplicare la firma digitale ad un documento con nuovi algoritmi sia in maniera periodica che in caso di necessità.

Andrea Pasquinucci

pasquinucci@ucci.it

Riferimenti Bibliografici

[1] Annuncio di Bruce Schneier sulla rottura di SHA-1:
http://www.schneier.com/blog/archives/2005/02/sha1_broken.html

[2] Annuncio dei ricercatori cinesi: http://theory.csail.mit.edu/~yiqun/shanote.pdf

[3] Una breve introduzione agli algoritmi di Hash: http://www.unixwiz.net/techtips/iguide-crypto-hashes.html