Brute-force Collision Search for (truncated) MD5

Andrew Isaacson
isaacson@cs.umn.edu

***Abstract*— A survey of the state of the art in distributed collision search and results of an implementation of same applied to truncated MD5.**

I. Introduction

The MD5 algorithm is "an extension of the MD4 message-digest algorithm" that is "slightly slower than MD4, but is more "conservative" in design." [1]. It is a cryptographic hash function with 128-bit output. The most significant publicly known algorithmic weakness in MD5 is the collision in its compression function [2].

The most significant design weakness in MD5 is the small size of its output. [3] showed in 1996 that for a $10 million budget a machine could be constructed to generate collisions in MD5 (or, indeed, nearly any hash function with 128-bit output) in an average 22 days. This is achieved by applying the "birthday paradox" to the collision search, resulting in a meet-in-the-middle (MITM) style of attack. Subsequent cryptographic hash functions have been specifed with 160-, 192-, 256-, or indeed 512-bit output, to make a MITM hash collision attack infeasible.

The modern cryptography community knows that MD5 is too small for comfort, and counsels against its use [4]. Even back in 1994, [5] suggested alternative hash functions with larger output.

Nonetheless, MD5 continues to be the most popular cryptographic hash function in the general programming community. A significant number of SSL certificates are signed using RSA-MD5 [6]. Apple's new software distribution system uses MD5 exclusively [13]. In January 2004, LiveJournal.com fielded a new password negotiation scheme [14]. Their software library included both MD5 and SHA-1; this protocol was being constructed anew out of whole cloth (there was no backwards compatibility consideration). Nonetheless, the designer specified MD5 rather than SHA-1.

There are some bright points. The OpenBSD project [8] provides three hashes (MD5, SHA-1, and RMD160) as fingerprints to ensure integrity of files downloaded via their 'ports' system.

Experience with DES and RC5 indicates that programmers and designers without extensive cryptographic experience will continue to ignore hypothetical arguments showing insecurity, until an actual break is demonstrated in practice. Ergo, an open demonstration of a collision in MD5 will likely have a salutary effect in moving forward the practicum of cryptography.

II. Background

A. Internet-distributed computing

There have been several successful efforts coordinating loosely-coupled groups of volunteers over the Internet to attack trivially parallelizable cryptography problems. The DES Challenge group, led by Rocke Verser, was one of the first. In 1997, RSA Laboratories offered $10,000 cash to anyone who could decrypt a message encrypted with a 56-bit DES key. Five months later, DESCHALL (as Rocke's group was known) claimed the prize. Ongoing challenges led to DES keys being broken in as little as 22 hours. [7], [9]

Since then, the Distributed.net team has attacked a variety of problems with great success. Tens of thousands of volunteers are currently donating cycles from hundreds of thousands of computers in pursuit of a variety of problems, both cryptographic and mathematical.

There are dozens of other distributed computing volunteer projects, including SETI@Home [10], Folding@Home [11], GIMPS [12], and a plethora of others.

B. collision techniques for hash functions

[3] describes a feasible technique to find collisions in a hash function such as MD5. The strategy is to define a "random walk" across the space of all MD5 hashes, with each iteration representing a hash of one of the messages at hand. (This is not really a random walk, because the iterator is a pure function of the previous point. Perhaps "deterministic walk" would be more accurate.) Eventually two 'threads' or 'streams' of iterated hashes will coincide, which can be detected by keeping a database of points (or hashes) visited. The storage required to hold these data can be mitigated by storing only a small percentage of the points visited; viz, a *distinguished point*, or hash with a prefix of n bits all zero. n can be tuned to the expected number of points to be computed; for full 128-bit MD5, $n = 32$ is reasonable, while a truncated 48-bit MD5 is well served by $n = 16$. We prefer to compute a large number of streams of shorter length rather than fewer extremely long streams, so that the percentage of initial collisions can be maximized (as opposed to two streams which merge and continue in parallel, resulting in a useless collision detected at every DP).

III. Design

The code is structured as follows. There is a *database* which stores (DP, stream, position) tuples. There is a *server* which collects tuples from multiple *clients* over a network. Finally, there is an *iterator* which computes the hash values and identifies distinguished points, which can be stored in a local

DB or fed through the client to a remote server and thence to a centralized DB. For short runs (for example, finding a 64-bit collision) a single machine with a local DB suffices; for longer runs (more than 72 bits) multiple client machines are necessary to avoid overly long runtimes.

A. Database

The database stores (hash, streamID, position) tuples. When mounting an attack on full 128-bit MD5, the database will be required to store 2^{32} tuples, and must answer the question "do any two tuples share the same hash?" The obvious solution is to store the tuples sorted by hash value.

The database format proposed in [3] stores 12 bytes of hash, 4 bytes to name streams, and 6 bytes to store iteration count; at 22 bytes per tuple, van Oorschot and Wiener calculate that their database will require 30 GB of memory. (The calculations are somewhat involved; consult the reference for details.) They suppose to spend $2.25 million on RAM. (Remember, it was 1996 – they estimated $75 per megabyte. $300 per gigabyte would be more accurate, as of mid-2004.) Instead, I chose to store my database on disk, using streaming techniques to mitigate the impact of slow random access times.

As tuples are received from the network, they are stored, sequentially, to a file on disk. Once enough tuples have been received to fill 75% of system RAM, the entire *queue* is mapped (`mmap`(2)) and sorted. The system then iterates through the list of tuples, storing them to the appropriate place in the actual database, which is streamed linearly through system memory in reasonably-sized blocks.

Using random access, with a working set larger than available cache, a modern disk can sustain on the order of 100-500 I/O operations per second (IOPS); a large and expensive RAID system might attain 10,000 IOPS. Due to the write-mostly nature of the tuple workload, and by deferring collision detection until queue processing time, this database system can sustain write rates of 18,000 tuples per second on a single inexpensive ATA/100 drive.

B. Network protocol

Each client opens a TCP connection to the server; the client and server then communicate using an SMTP-inspired ASCII protocol. Under the current protocol, the client sends each DP to the sever as an independent transaction; an obvious optimization would be to batch them to minimize TCP and IP overhead.

The server uses a polling event loop implemented with `select`(2) and non-blocking IO to avoid the difficulties of managing multi-process access to a single file.

C. The hash iterator function

First, we assume that there are two messages, m and m'; the goal is to find perturbations of m and m' such that the perturbed messages hash to the same value under MD5. A hash value with its first 32 bits all zero is a *distinguished point* (or more precisely, a 32-bit DP).

Let $H : \{0,1\}^n \rightarrow \{0,1\}^{128}$ represent MD5, and $h \in \{0,1\}^{128}$ a particular instance of an MD5 hash. A perturbation function $f_m(h)$ (respectively, $f_{m'}(h)$) takes a 128-bit input h and outputs m_h semantically identical to m and uniquely determined by h. We choose a binary predicate $p(h)$ that is easily computable, such as parity. A choice function $e(h)$ is defined

$$e(h) = \begin{cases} f_m(h) & \text{if } p(h) = 0 \\ f_{m'}(h) & \text{if } p(h) = 1 \end{cases}$$

Finally, we construct a *hash iteration function* $g : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$

$$g(h) = H(e(h))$$

The messages m and m' should be of the same length, and the perturbation function should be length-preserving. (While it is possible to implement the collision technique given messages of differing lengths, this precludes some compelling optimizations.) A simple example of a perturbation function is an HTML document with a hidden string such as `<a href="AAAAAAAA"></A>`; the A's can be replaced by any alphanumeric string without any visible or semantic change. Another permutation applicable to source code and plaintext files is trailing spaces between the last printable character and the carriage return terminating the line. HTML also allows arbitrary whitespace to be substituted for other whitespace.

The perturbation functions for m and m' do not need to be related.

Clearly, given an unbiased $p(h)$ and $H()$, there will be approximately equal numbers of $H(f_m())$ and $H(f_{m'}())$ generated. In any particular collision, the probability is $p = 0.5$ that the collision is between m and m'. The birthday "paradox" assures us that once we approach $\sqrt{|H|}$ hashes generated, we will quickly generate a large number of collisions; eventually, a desired collision between $H(f_m())$ and $H(f_{m'}())$ will be found.

As noted above, we prefer to have many streams of shorter length rather than fewer, longer streams. Therefore, the iterator restarts with a new stream after 100 DPs are found.

IV. Implementation

[15] contains an implementation of this design in about 1700 lines of C, written to the Unix API (libc + mmap + sockets) with OpenSSL providing the cryptographic primitives.

V. Performance

Using OpenSSL 0.9.7c on an "Athlon XP 2400+" running at 2 GHz, with a message 178 bytes long, the iterator is able to compute 1.1 million iterations per second. In this implementation, speed is closely related to the length of the message, due to OpenSSL's implementation.

VI. Results on truncated MD5

Running on the aforementioned Athlon XP for one hour with HASHBIT=64, the code completed 4.06 billion iterations, finding 61,874 distinguished points, at a rate of 1.11 million iterations per second. It found one 64-bit collision: the message

```
<html><head>
```

```
<title>Orders from your Captain</title>
</head><body>
String Jack Shaftoe up by his thumbs.
<br>
<i>Cap'n Hook</i>
</body></html>
<AhadDagACAbHGafaAAAAAAAAAAAAAAAA>
```

hashes to 1ec2052f43a150cbd90b02e85267e99e while the message

```
<html><head>
<title>Orders from your Captain</title>
</head><body>
Pay Jack Shaftoe 100 pieces of eight.
<br>
<i>Cap'n Hook</i>
</body></html>
<GddGffbedAeaHCDFAAAAAAAAAAAAAAAA>
```

hashes to 1ec2052f43a150cb2e1b1bb64a77c63b. The preceding DPs were p1=00001a90d2021a0c00000000 and p2=00005c5a40350f8a00000000, and preceding hash values were z1=f071160d04e31c1b0000000000000000 and z2=7cc7bb9307194ea60000000000000000. The collision came 186,487 iterations after the closest DP.

VII. Applications to full MD5

At 1 million iterations per second, 2^{64} iterations (expected to find a collision in a 128-bit hash function) would require 584,000 CPU-years, or 20,000 CPUs for 29 years, or 100,000 CPUs for 5.8 years. At 11 million iterations per second, the expected CPU time goes down to 53,000 CPU-years, or 100,000 CPUs for 6.3 months. This is somewhat larger than most of the successful distibuted computing projects, but is much smaller than the CPU-years consumed by the RC5-64 project of Distributed.net, thanks to Moore's Law.

VIII. Future Work

This is a very active field and there are several fruitful areas of ongoing work. Here are a few ideas that promise immediate reward.

A. Avoid computing the full hash

The MD5 alorithm processes its input as a series of 512-bit blocks, with the length of the input included in a final "padding" block. The previous state, S_{i-1}, is mixed with the current block, B_i:

$$S_i = f(S_{i-1}, B_i)$$

If m and m' are the same length and differ only in block B_i, then $S_{i-1} = S'_{i-1}$, and finding a collision in MD5 reduces to finding a collision beween $f_{S_{i-1}}(B_i)$ and $f_{S_{i-1}}(B'_i)$.

To the best of my knowledge, none of the current crop of software MD5 collision searchers implement this optimization.

B. Compute multiple streams using explicit paralellism

Explicit parallelism involves performing the same operation on many data with a single instruction. Modern CPUs provide a variety of implementations of this basic idea, including MMX, SSE, SSE2, and AltiVec (even ignoring the much wider parallelism available on vector machines such as the Cray X1). Furthermore, even using plain 32-bit ALU operations it can be fruitful to "turn the problem on its side" and compute 32 one-bit operations simultaneously. DESCHALL called this approach "bit-slicing" and recognized significant speedups (on the order of 4-10x) from it.

The `md5crk.com` group has an implementation using SSE2, and reports that it performs about 10x better than a plain C implementation on a Pentium 4 at 2.4 GHz.

C. Programmable Hardware

Key search can be very efficiently implemented in an ASIC; as the EFF's DES Cracker showed, this is feasible at a suprisingly low price point (under $300,000). An even lower price-point is available with modern FPGA hardware, which frequently include a hard CPU and Ethernet transceiver on-chip.

D. Programmable GPUs

Modern GPUs such as the ATI Radeon and NVidia NV40 have an extremely flexible programming interface, with a rich set of operators designed to be applied to large vertex lists and pixel arrays, and with extremely high parallelism. Some researchers are already running LINPACK on such GPUs; it seems likely that MD5 could be implemented efficiently as well.

IX. Conclusion

Clearly, the $10 million budget envisioned in [3] has fallen to Moore's Law. Today, that 30GB of RAM which cost $2.25 million could be had, with a Gigabit Ethernet interconnect, for under $12,000.

Despite the likelihood that various governmental and commercial organizations have constructed such a machine, no acknowledgement of a true MD5 collision has yet been published. Clearly the software and database capability exists; the only remaining hurdle is the relative slowness of software implementations.

References

[1] RFC 1321, `http://www.ietf.org/rfc/rfc1321.txt`
[2] *Cryptanalysis of MD5 Compress*, H. Dobbertin, May 2, 1996
[3] *Parallel Collision Search with Cryptanalytic Applications*, P. van Oorschot and M. Wiener, 1996
[4] Schneier and Ferguson, *Practical Cryptography*
[5] Schneier, *Applied Cryptography*
[6] `http://www.md5crk.com/?sec=howinsecure`
[7] *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, EFF
[8] `http://www.openbsd.org/`
[9] `http://www.distributed.net/des/`
[10] `http://setiathome.ssl.berkeley.edu/`
[11] `http://www.stanford.edu/group/pandegroup/folding/`
[12] `http://www.mersenne.org/prime.htm`
[13] `http://www.vmeng.com/pipermail/mac_crypto/2004-April/000645`

[14] http://www.livejournal.com/community/lj_dev/599743.html
[15] http://web.hexapodia.org/~adi/md5dp/