MD5 چیست؟

الف) یک روش رمزنگاری　　ب)یک روش رمزگشایی　　ج) یک روش حمله　　د) گزینه الف و ب

MD5 (Message Digest)

۵ گام برای پیاده سازی و طرز کار الگوریتم MD5

استاد درس: مهندس سمانه ایرانیان

دانشجو: محمد جواد ملکی

شماره دانشجویی: ۹۲۲۱۱۷۰۰۲۱

درس امینت شبکه

دانشگاه کاشان

۱۳۹۵

مقدمه

MD5 ، یک روش رمزنگاری است که به صورت گسترده به عنوان تابع درهم‌ساز رمزنگارانه استفاده می‌شود. این الگوریتم یک رشته با طول متفاوت را به عنوان ورودی می‌گیرد و یک خلاصه پیام امدی۵ یا اثر انگشت با طول ۱۲۸بیت می‌سازد. الگوریتم امدی۵ توسعه‌ای از الگوریتم امدی۴ است با این تفاوت که‌امدی۵ کمی کندتر از امدی۴ عمل می‌کند اما در طراحی آن بسیار محافظه کارانه عمل شده‌است.

امدی۵ در شرایطی طراحی شد که حس کردند امدی۴ به علّت سرعت بالایی که دارد پذیرفته شده‌اما از امنیت مناسبی در شرایط بحرانی برخوردار نیست. امدی۵ کمی نسبت به‌امدی۴ کندتر شد، درعوض امنیت آن بیشتر گشت. این الگوریتم حاصل تأثیر دادن نظرات تعدادی از استفاده کنندگان امدی۴ به همراه مقادیری تغییر در ساختار الگوریتم برای افزایش سرعت و قدرت آن می‌باشد.

5 گام برای پیاده سازی به این صورت در ادامه بیان خواهند شد:

گام 1- اضافه کردن بیتهای نرم کننده

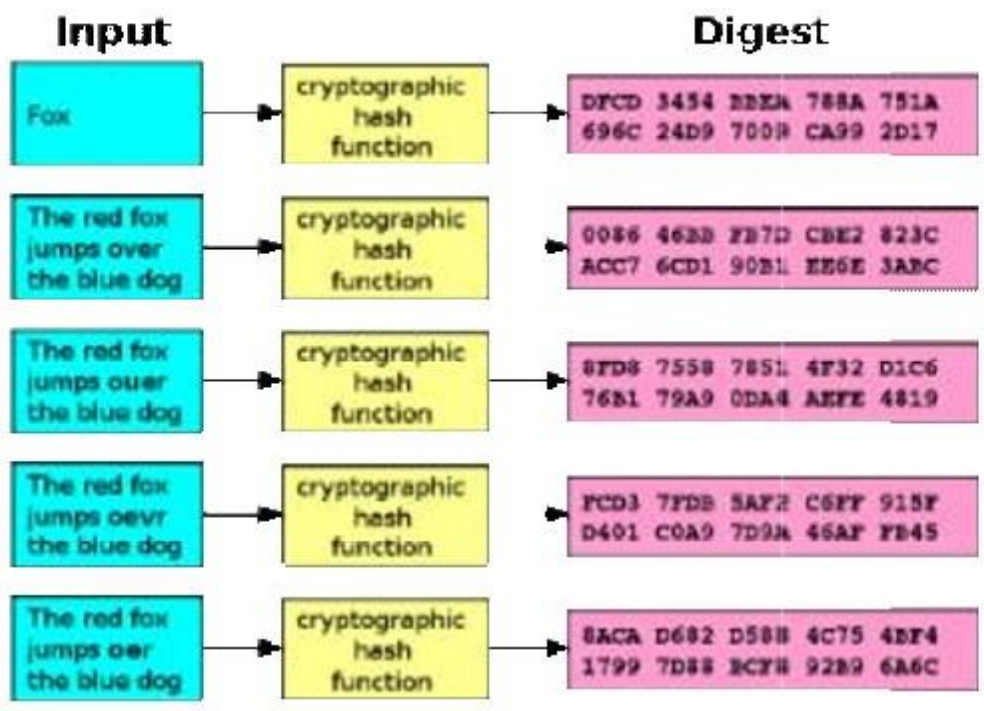
گام 2- افزایش طول

گام 3- بین بافر برایMD

گام 4- پردازش پیام در بلاک های 16 کلمه ای

گام 5- خروجی

MD5که مخفف عبارت Message Digest algorithm 5 یا الگوریتم شماره ۵ خلاصه پیام است، در سال ۱۹۹۱ توسط یکی از حرفه ای ها و سرآمدان علم رمزنگاری آمریکا، پروفسور رونالد ریوست اختراع گردید و جایگزین استاندارد قدیمی MD4 شد . MD5نام ساده ای برای نوعی از رمزنگاری و عملیات پنهان سازی است که رونالد از سال ۹۱ آغاز کرد.



ایده پشت درهم و برهم سازی داده ها برای پنهان سازی آنها یا cryptographic hashing، اینگونه است که یک بلوک قراردادی از داده ها گرفته شده و در پاسخ یک مقدار هش شده با اندازه مشخص برگردانده می شود. که این داده می تواند هر نوع اطلاعاتی باشد. و هر اندازه ای داشته باشد. اما داده هش شده همیشه اندازه و مقدار مشخص و ثابتی دارد. برای مثال می توانید در اینجا یک برنامه هش کننده MD5 را امتحان کنید و هر نوع داده مانند عدد و متن را که دوست دارید تبدیل به یک هش ام دی ۵ کنید.

پنهان سازی با هش کردن کاربردهای متعددی می تواند داشته باشد و الگوریتم های مختلفی برای این کار وجود دارد (که ام دی ۵ یکی از آنها است) که کار نسبتا یکسانی انجام می دهند. یکی از کاربردهای رمزنگاری با هش کردن، تایید محتوای پیام ها یا فایل ها پس از انتقال آنها است.

اگر شما یک فایل نسبتا بزرگ را دانلود کنید، معمولا یک مقدار هش شده را هم خواهید یافت که در فایل جداگانه ای این فایل را همراهی می کند. یک بار که این فایل دانلود شود، می توانید کنترل کنید که فایل دانلود شده با فایل اصلی مطابقت کامل داشته و بدون هیچ تغییری به دست شما رسیده باشد.

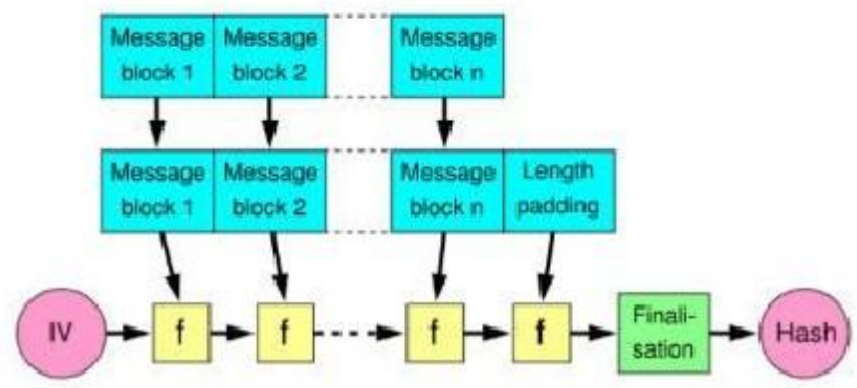
همین شیوه برای پیام ها هم کاربرد دارد. با استفاده از هش می توانید بازبینی کنید که پیام دریافتی با پیام ارسالی کاملا یکسان باشد. در سطوح اولیه کاربرد، اگر شما و دوست تان دو فایل خیلی حجیم دارید و می خواهید بدون جابجایی حجم بالای اطلاعات از طریق اینترنت، مطمئن شوید که هر دو فایل یکسان هستند. این شیوه به کمک تان خواهد آمد.

الگوریتم های هش کردن، همچنین به عنوان بخشی از دنیای تایید هویت فایل و اطلاعات نقش بازی می کنند. یک مثال خوب برای این مورد، شبکه های اشتراک فایل نظیر به نظیر مانند eDonkey و تورنت هستند. سیستم ای دانکی از انواعی از الگوریتم های ام دی ۴ به همراه حجم فایل مورد نظر، برای هش کردن استفاده می کند تا فایل هایش را در سیستم علامت گذاری کند.

یکی دیگر از استفاده های هش ها، نگه داری و ذخیره رمزهای عبور است. به دلایل کاملا آشکاری، دسته بندی و ذخیره کردن رمزهای عبور به صورت متن ساده شیوه بسیار بد و خطرناکی است (مثلا وقتی هکر به بانک اطلاعاتی سایت با رمزهای ذخیره شده به صورت متن ساده دست پیدا کند، بدون هیچ دردسری علاوه بر تمام اطلاعات حساب هر کاربر، به رمز مشترک بسیاری از حساب های کاربر در سایت های دیگر هم دست یافته!). پس آنها را به عبارات هش شده تبدیل می کنند. وقتی کاربری رمزعبور را وارد می کند، سیستم آن را به یک عبارت هش تبدیل می کند، و سپس آن را با مقدار هش ذخیره شده معتبر مقایسه می کند. از آنجایی که عملیات هش کردن یک فرآیند یک طرفه است، از نظر تئوری شانس بسیار کمی وجود دارد که رمزعبور اصلی از دل یک عبارت هش شده فاش شود.

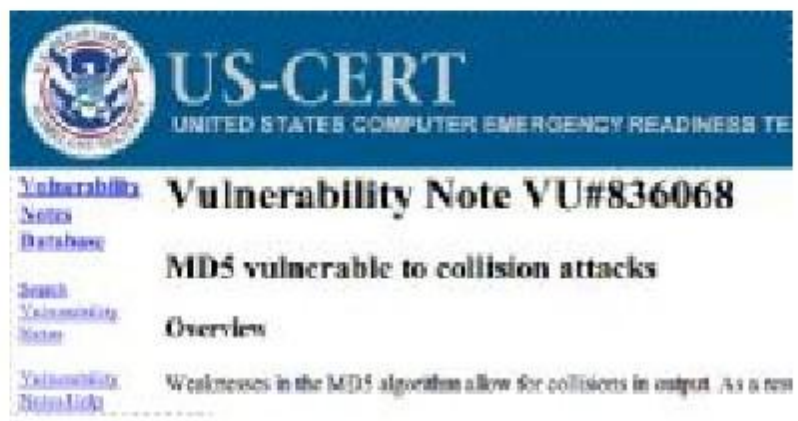
الگوریتم شماره ۵ گوارش پیام

تابع MD5 یک عدد هگزادیسیمال ۳۲ رقمی را فراهم می سازد. اگر ما negahbaan.com را به مقدار ام دی ۵ آن تبدیل کنیم، چیزی شبیه f960c492f10646c23ef750711709a36۶ خواهیم داشت. این مقدار بر پایه شیوه ای به نام ساختار Merkle–Damgård (عکس پایین) تولید می شود که بیشتر با نام توابع هش ضد تصادم (collision-proof) شناخته می شوند.



البته هیچ گاه نمی توان چیزی پیدا کرد که در برابر همه حملات کاملا امن باشد. همانطور که در سال ۱۹۹۶ رخنه های بالقوه ای در الگوریتم هش کردن MD5 هم کشف شد. البته این مشکلات کشنده نبودند و ام دی ۵ همچنان به کار خود ادامه داده و مورد استفاده قرار می گیرد. در سال ۲۰۰۴ هم بعد از تحقیقات گروهی از محققان که شرح می دادند چگونه دو فایل کاملا مجزا دارای یک هش ام دی ۵ کاملا یکسانی هستند، کارشناسان متوجه یک ایراد امنیتی دیگر در این الگوریتم شدند. این مورد اولین شاهد از حمله تصادم بود که بر علیه الگوریتم ام دی ۵ به کار می رفت. یک حمله تصادم که کوشش می کرد دو خروجی مجزا را بیابد که دارای یک ارزش هش یکسان باشند.

بعد از چند سال، تلاش برای یافتن مشکلات امنیتی دیگر در ام دی ۵ باز هم نتایجی به همراه داشت. در سال ۲۰۰۸ یک تحقیق گروهی دیگر برای استفاده از شیوه حمله تصادم جهت تقلب در اعتبار سنجی مجوز SSL ترتیب داده شد. با این کار باید کاربران ساده لوح خیال می کردند که در حال وبگردی امن هستند، در حالی که این گونه نبود. اداره امنیت سرزمینی آمریکا اخطار کرده که: «کاربران باید از به کار بردن MD5 در هر کاری اجتناب کنند. همانگونه که تحقیقات قبلی هم این امر را ثابت کرده است. این باید کاملا و به صورت واضح اعلام گردد که این شیوه رمزنگاری شکسته شده است و برای استفاده بیشتر نامناسب است.»


US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TE
Vulnerability Note VU#836068
MD5 vulnerable to collision attacks
Overview
Weaknesses in the MD5 algorithm allow for collisions in output. As a res

علی رغم اخطار دولت آمریکا، بسیاری از سرویس های درون این کشور و دیگر نقاط دنیا هنوز از MD5 استفاده می کنند و هر چند که از نظر تکنیکی و فنی کاملا در معرض خطر قرار داشته باشند. هرچند که هنوز امکان نمک سود! (salt) کردن رمزهای عبور را هم برای جلوگیری از حملات فرهنگ نامه ای داریم. اگر یک هکر لیستی از رمزهای عبوری که به صورت تصادفی بیشتر استفاده می شوند را داشته باشد و همچنین نام کاربری اکانت دیتا بیس سایت شما را هم پیدا کند، می تواند به راحتی لیستی از هش های رمزهای عبور مشهور را برای ورود تست کند. Salt یک String یا رشته تصادفی است که به رمزعبور هش شده لینک می گردد و مجددا هش می شود. آنگاه مقدار سالت و نتایج هش در دیتا بیس ذخیره می گردند.

اگر یک هکر بخواهد به رمزهای عبور مرتبط با نام های کاربری شما دست پیدا کند، ابتدا باید هش های سالت را کشف کند که برای این کار حمله فرهنگ لغات تقریبا غیر قابل استفاده است.

MD5 یکی از شیوه های بسیار متنوع تایید هویت، امن سازی و بازبینی اطلاعات است. هش کردن رمزنگاری شده یک بخش حیاتی در تاریخ امنیت است و همه چیز را مخفی نگه می دارد. اما همانند همه چیزهایی که برای امنیت ساخته شده اند و باطن آنها بر امنیت پایه گذاری شده، این هم ممکن است شکسته شود.

شما احتمالا لازم نیست که زیاد درباره هش کردن و کنترل مقابله ای MD5 در وبگردی و زندگی آنلاین روزمره تان نگران باشید. اما حداقل هم اکنون می دانید که آنها چه کار می کنند و چگونه این کار انجام می شود

شرایط و لوازم کار جهت پیاده سازی

در این متن منظور از «کلمه» تعداد ۳۲ بیت و «بایت» تعداد ۸ بیت داده می‌باشد. یک صف از بیت‌ها دارای خصوصیات طبیعی یک صف از بایت‌ها می‌باشند که هر گروه هشت تایی متوالی از بیت‌ها یک بایت را تشکیل می‌دهند که پرارزش‌ترین بیت در ابتدا قرار دارد. یک صف از بایت‌ها دقیقا مشابه یک صف ۳۲ بیتی از کلمات پردازش می‌شود. جایی که گروهی ۴ تایی از توالی بایت‌ها پردازش می‌شوند، کم ارزش‌ترین بایت اولین بایت می‌باشد.

اجازه بدهید از x_i بجای x (xi اندیس i) استفاده کنیم و اگر مقدار اندیس یک عبارت محاسباتی بود آن را در {} محدود می‌کنیم، مانند: x_{i-1}. همچنین از ^ به عنوان علامت توان استفاده می‌کنیم، پس x^i یعنی x به توان i.

اجازه بدهید از علامت «+» برای اضافه کردن دو کلمه به هم استفاده کنیم. از x<<<5 به عنوان عملگر چرخش بیتی در کلمات استفاده می‌شود که x به‌اندازه ۵ بیت به چپ چرخش می‌کند.

از not(x) به عنوان عملگر نقیض بیتی، از X v Y به عنوان عملگر فصل (or) و از X xor Y به عنوان عملگر exclusive or و از XY به عنوان عملگر عطف (and) استفاده می‌کنیم.

توضیحات الگوریتم MD5

فرض کنید ما b بیت پیام به عنوان ورودی داریم و تصمیم داریم خلاصه پیام آن را بدست آوریم. b در اینجا یک عدد نا منفی و صحیح است، b می تواند مقدار صفر داشته باشد و هیچ محدودیتی برای مضرب هشت بودن آن نیست و به هر اندازه می تواند بزرگ باشد. فرض کنید بیت‌های این پیام را بشود به صورت زیر نوشت:

m_0 m_1... m_{b-1}

$m_0 m_1 \dots m_{b-1}$

برای آوردن خلاصه پیام ۵ مرحله زیر را انجام می‌دهیم.

1::::اضافه کردن بیتهای نرم کننده

طول پیام مورد نظر به ۴۴۸ به پیمانه ۵۱۲ توسعه پیدا می کند به این معنی که اگر به طول پیام ۶۴ بیت اضافه شود، طولش مضربی از ۵۱۲ خواهد بود. عمل توسعه دادن همیشه اجرا می‌شود مگر اینکه طول پیام به صورت ۴۴۸ به پیمانه ۵۱۲ باشد.

عمل توسعه پیام یا نرم کردن آن به صورت زیر انجام می‌شود:

یک بیت [۱] سپس تعدادی بیت [۰] به پیام اضافه می‌شود. اضافه شدن بیت‌های ۰ تا زمانی که طول رشته به ۴۴۸ بر پایه ۵۱۲ برسد، ادامه پیدا می‌کند. در این عمل حداقل یک بیت و حداکثر ۵۱۲ بیت اضافه خواهد شد.

2::::افزایش طول

یک نمایش ۶۴ بیتی از b بیت پیام اولیه به آخر نتیجه گام قبل اضافه می‌شود. در بدترین حالت، b بزرگ‌تر از ۶۴ بیت خواهد بود. در این حالت فقط ۶۴ بیت کم ارزش b استفاده خواهد شد.

هم اکنون طول پیام بدست آمده دقیقا معادل مضربی از ۵۱۲ خواهد بود. مشابه اینکه بگوییم، این پیام طولی معادل مضربی از۱۶ کلمه دارد اجازه بدهید M[0...N-1] را نمایانگر کلمات پیام بدست آمده بدانیم. (N مضربی از ۱۶ می‌باشد.)

3::::تعیین بافر برای MD

برای محاسبه خلاصه پیام یک بافر ۴ کلمه‌ای (A,B,C,D) استفاده می‌شود. هر کدام از A، B، Cو D یک ثبات ۳۲ بیتی می‌باشند. این ثبات‌ها مطابق جدول زیر مقدار دهی می‌شوند (بایتهای کم ارزش در ابتدا قرار دارند)

word A: 01 23 45 67

$wordA : 01234567$

word B: 89 ab cd ef

$wordB : 89abcdef$

word C: fe dc ba 98

$wordC : fedcba98$

word D: 76 54 32 10

$wordD : 76543210$

4::::پردازش پیام در بلاک‌های ۱۶ کلمه‌ای

در ابتدا ۴ تابع کمکی تعریف می‌کنیم که هر کدام به عنوان ورودی سه کلمه ۳۲ بیتی می‌گیرد و برای خروجی یک کلمه ۳۲ بیتی تولید می‌کند.

F(X,Y,Z) = XY v not(X) Z

$F(X,Y,Z) = XY v not(X) Z$

G(X,Y,Z) = XZ v Y not(Z)

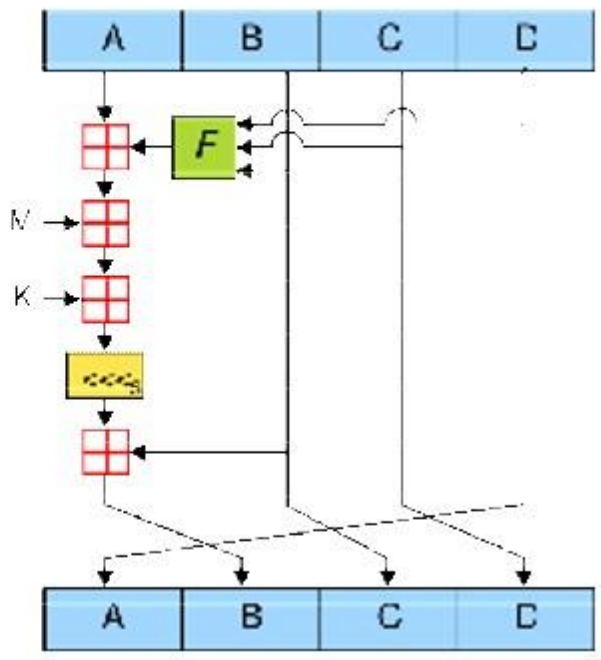
$G(X,Y,Z) = XZ v Y not(Z)$

H(X,Y,Z) = X xor Y xor Z

$H(X,Y,Z) = X xor Y xor Z$

I(X,Y,Z) = Y xor (X v not(Z))

$I(X,Y,Z) = Y xor (X v not(Z))$



$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations respectively.

در هر موقعیت بیتی، F به عنوان شرط عمل می کند: اگر X آنگاه Y در غیر این صورت Z. تابع F می توانست طوری تعریف شود که به جای استفاده از v از + استفاده کند چون XY و not(X) هرگز یک هایی در موقعیت بیتی یکسان نخواهد داشت. جالب است به یاد داشته باشید که اگر بیت‌های X، Y و Z مستقل و غیر مرتبط باشند، هر بیت از F(X, Y, Z) مستقل و غیر مرتبط خواهد بود.

توابع G، H و I شبیه تابع F هستند، به طوری که آنها در "توازی بیتی" کار می کنند تا خروجی شان را از بیت های X، Y و Z تولید کنند. در چنین روشی اگر بیت‌های متناظر X، Y و Z مستقل و غیر مرتبط باشند، آنگاه هر بیت از (G(X, Y, Z، H(X, Y, Z و I(X, Y, Z) مستقل و غیر مرتبط خواهند بود.

توجه داشته باشید که تابع H، تابع XOR یا توازن بیتی از ورودی هایش است. این گام از یک جدول ۶۴ عنصری T[1...64] ساخته شده از یک تابع مثلثاتی، استفاده می کند. اجازه دهید T[i] را که i-امین عنصر جدول را مشخص می کند که برابر است با قسمت صحیح حاصلضرب ۴۲۹۴۹۶۷۲۹۶ در abs(sin(i))، به طوری که i به رادیان باشد.

5:::: خروجی:

خلاصه پیامی که به عنوان خروجی تولید می شود و عبارت است از A، B، C و D، که ما با کم ارزش ترین بیت A شروع می کنیم و به با ارزش ترین بیت D خاتمه می دهیم. این تعریف MD5 را کامل می کند.

الگوریتم خلاصه پیام MD5 به سادگی قابل اجرا می باشد و یک "اثر انگشت" یا "خلاصه پیام" از پیام با طول اختیاری تولید می کند. گمان برده می شود که امکان مواجه شدن با دو پیام که خلاصه پیام مشابهی دارند از رتبهء 2^64 و برای هر پیامی که به آن یک خلاصه پیام داده شده است از رتبهء 2^128 می باشد.

الگوریتم MD5 برای نقاط ضعف به دقت بررسی شده است. به هر حال این الگوریتم نسبتا جدید است و تحلیل امنیتی بیشتری را طلب می کند، مشابه طرح های مشابه در این رده.

نحوه تبدیل یک متن ساده به MD5

```
/* پردازش هر بلاك 16 كلمه اي. */
For i = 0 to N/16-1 do
        /* كپي هر بلاك كلمه */
        /* i into X. */
        For j = 0 to 15 do
                Set X[ j] to M[ i*16+j] .
        end /* of loop on j */
        /* Save A as AA, B as BB, C as CC, and D as DD. */
        AA = A
        BB = B
        CC = C
        DD = D
        /************************** گام اول **************************/
        /* Let [ abcd k s i] denote the operation
        a = b + ((a + F(b,c,d) + X[ k] + T[ i]) <<< s). */
        /* Do the following 16 operations. */
        [ ABCD  0  7  1]  [ DABC  1 12  2]  [ CDAB  2 17  3]  [ BCDA  3 22  4]
        [ ABCD  4  7  5]  [ DABC  5 12  6]  [ CDAB  6 17  7]  [ BCDA  7 22  8]
        [ ABCD  8  7  9]  [ DABC  9 12 10]  [ CDAB 10 17 11]  [ BCDA 11 22 12]
        [ ABCD 12  7 13]  [ DABC 13 12 14]  [ CDAB 14 17 15]  [ BCDA 15 22 16]
        /************************** گام دوم **************************/
        /* Let [ abcd k s i] denote the operation
        a = b + ((a + G(b,c,d) + X[ k] + T[ i]) <<< s). */
        /* Do the following 16 operations. */
        [ ABCD  1  5 17]  [ DABC  6  9 18]  [ CDAB 11 14 19]  [ BCDA  0 20 20]
        [ ABCD  5  5 21]  [ DABC 10  9 22]  [ CDAB 15 14 23]  [ BCDA  4 20 24]
        [ ABCD  9  5 25]  [ DABC 14  9 26]  [ CDAB  3 14 27]  [ BCDA  8 20 28]
        [ ABCD 13  5 29]  [ DABC  2  9 30]  [ CDAB  7 14 31]  [ BCDA 12 20 32]
        /************************** گام سوم **************************/
        /* Let [ abcd k s t] denote the operation
        a = b + ((a + H(b,c,d) + X[ k] + T[ i]) <<< s). */
```

```
/* Do the following 16 operations. */
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]
/**************************گام چهارم**************************/
/* Let [abcd k s t] denote the operation
     a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
/**************************گام پنجم**************************/
/*افزایش هر یک از چهار ثبات با مقدار آن قبل از اینکه بلاک آغاز شود*/
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* پایان حلقه */
```

خروجی

خلاصه پیامی که به عنوان خروجی تولید می‌شود و عبارت است از A ، B، CوD ، که ما با کم ارزش‌ترین بیت A شروع می‌کنیم و به با ارزش‌ترین بیت Dخاتمه می‌دهیم. این تعریف MD5 را کامل می‌کند .

نتیجه

الگوریتم خلاصه پیام MD5 به سادگی قابل اجرا می‌باشد و یک "اثر انگشت" یا "خلاصه پیام" از پیام با طول اختیاری تولید می‌کند. گمان برده می‌شود که امکان مواجه شدن با دو پیام که خلاصه پیام مشابهی دارند از رتبهٔ ۲^۶۴ و برای هر پیامی که به آن یک خلاصه پیام داده شده‌است از رتبه ۲^۱۲۸ می‌باشد .

الگوریتم * MD5 برای نقاط ضعف به دقت بررسی شده‌است. به هر حال این الگوریتم نسبتاً جدید است و تحلیل امنیتی بیشتری را طلب می‌کند، مشابه طرح‌های مشابه در این رده .

یک روش پیاده سازی دیگر با استفاده از همان فرمول بیان شده در بالا

```
//همه متغیرها 32 بیتی بدون علامت و بسته بندی پیمانه 2 ^ 32 در هنگام محاسبه
var int[ 64]  s, K

//s مقدار تغییر در هر دور مشخص
s[  0..15]  := {  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
s[ 16..31]  := {  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
s[ 32..47]  := {  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
s[ 48..63]  := {  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

// استفاده از باینری قسمت صحیح سینوس ها از اعداد صحیح (رادیان) را به عنوان
ثابت:
for i from 0 to 63
    K[ i]  := floor(abs(sin(i + 1)) × (2 pow 32))
end for
//(یا فقط با استفاده از جدول زیر):
K[  0.. 3]  := {  0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[  4.. 7]  := {  0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[  8..11]  := {  0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[ 12..15]  := {  0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[ 16..19]  := {  0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[ 20..23]  := {  0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[ 24..27]  := {  0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[ 28..31]  := {  0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[ 32..35]  := {  0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[ 36..39]  := {  0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[ 40..43]  := {  0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[ 44..47]  := {  0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[ 48..51]  := {  0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[ 52..55]  := {  0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[ 56..59]  := {  0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[ 60..63]  := {  0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

//مقداردهی اولیه متغیر :
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D

//قبل از پردازش: اضافه کردن یک 1 بیت
append "1" bit to message
/*  توجه: بایت ورودی به عنوان رشته بیت در نظر گرفته شده، که در آن اولین بیت
بیت کم ارزش ترین بایت است.[37]

//قبل از پردازش: بالشتک با صفر
append "0" bit until message length in bit ≡ 448 (mod 512)
append length mod (2 pow 64) to message

//پردازش پیام در تکه های 512 بیتی های پی در پی :
for each 512-bit chunk of message
    break chunk into sixteen 32-bit words M[ j], 0 ≤ j ≤ 15
//مقداردهی اولیه ارزش هش برای این تکه :
    var int A := a0
    var int B := b0
```

```
    var int C := c0
    var int D := d0
//**************************حلقه اصلی**********************:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            F := C xor (B or (not D))
            g := (7×i) mod 16
        dTemp := D
            D := C
            C := B
            B := B + leftrotate((A + F + K[ i]  + M[ g]), s[ i])
            A := dTemp
    end for
//اضافه کردن هش این تکه به نتیجه تا کنون :
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[ 16]  := a0 append b0 append c0 append d0
// (نیز little-endian خروجی در)
// تعریف تابع leftrotate
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

Note: Instead of the formulation from the original RFC 1321 above, the following may be used for improved efficiency (source: if assembly language is being used; otherwise, the compiler will generally optimize the above code. Since each computation is dependent on another in these formulations, this is often slower than the above method where the additions may be parallelized):

```
[ 0 ≤ i ≤ 15]: F :=  D xor (B and (C xor D))
[16 ≤ i ≤ 31]: F := C xor (D and (B xor C))
```

MD5 hashes [edit]

The 128 bit (16 byte) MD5 hashes (also termed message digests) are typically represented as a sequence of 32 hexadecimal digits. The following demonstrates a 43 byte ASCII input and the corresponding MD5 hash:

```
MD5("The quick brown fox jumps over the lazy dog")
  9e107d9d372bb6826bd81d3542a419d6
```

Even a small change in the message will (with overwhelming probability) result in a mostly different hash, due to the avalanche effect. For example, adding a period to the end of the sentence:

```
MD5("The quick brown fox jumps over the lazy dog.")
  e4d909c290d0fb1ca068ffaddf22cbd0
```

The hash of the zero-length string is:

```
MD5("")
= d41d8cd98f00b204e9800998ecf8427e
```

The MD5 algorithm is specified for messages consisting of any number of bits; it is not limited to multiples of eight bits (octets, bytes) as shown in the examples above. Some MD5 implementations such as md5sum might be limited to octets, or they might not support streaming for messages of an initially undetermined length.

یک روش پیاده سازی ساده به زبان سی

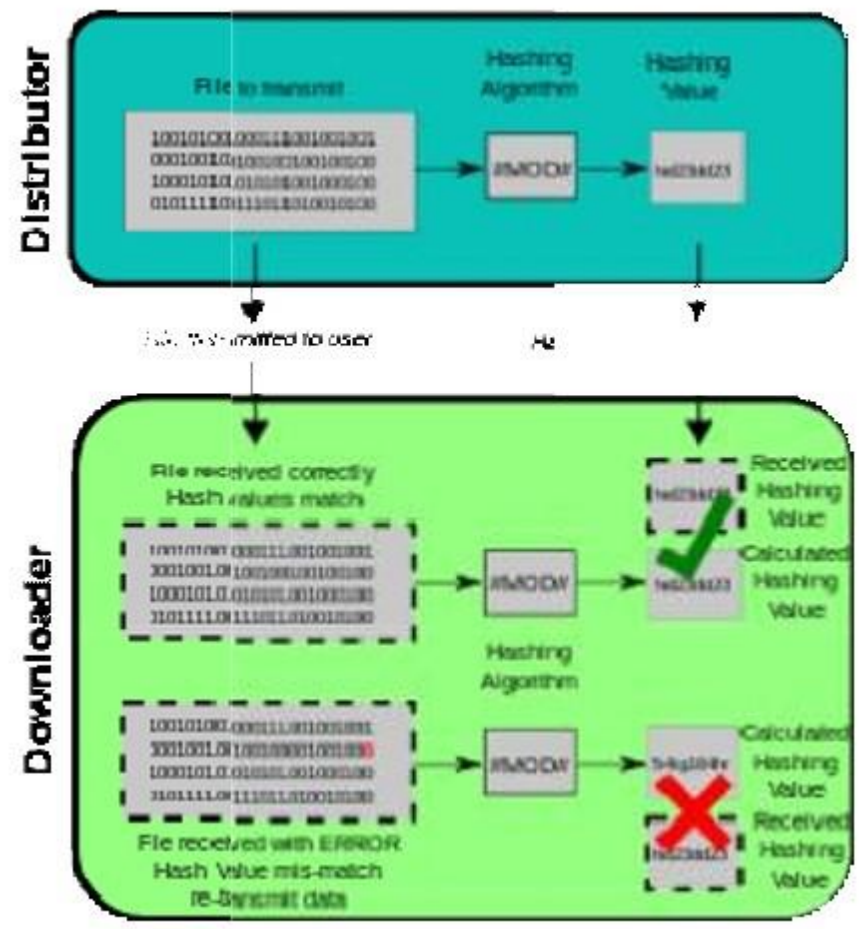
```
/*
 * Simple MD5 implementation
 *
 * Compile with: gcc -o md5 -O3 -lm md5.c
```

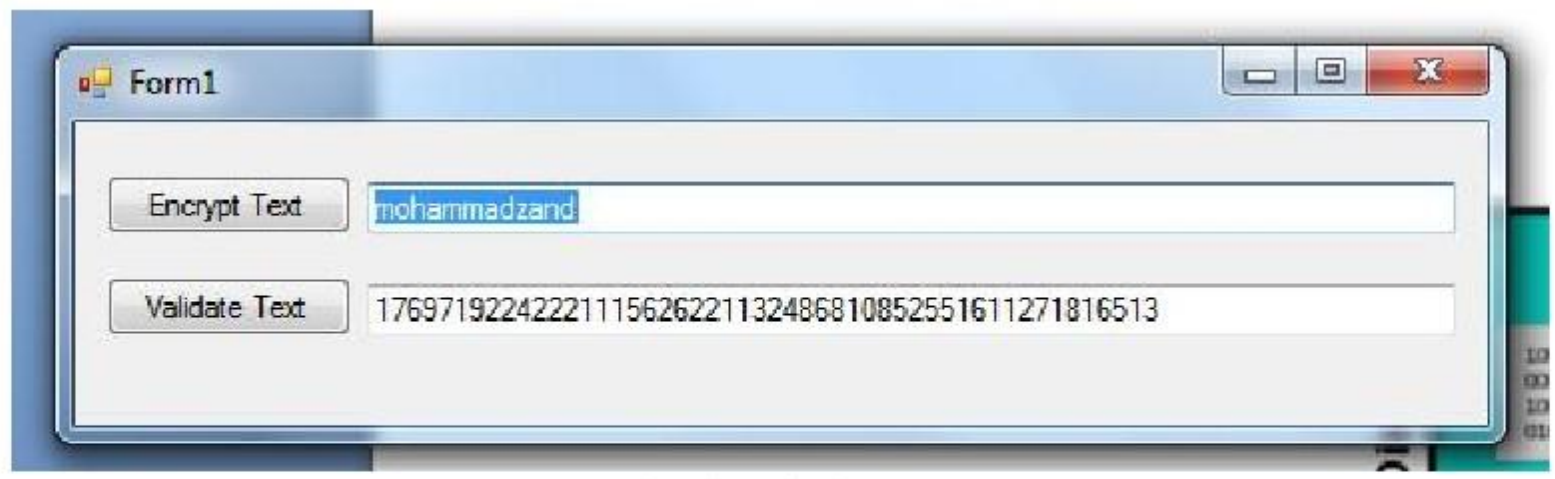
```
 *
 * NOTE: this code only works on little-endian machines.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

// Constants are the integer part of the sines of integers (in radians) *
2^32.
const uint32_t k[64] = {
0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,
0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,
0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

// leftrotate function definition
#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

// These vars will contain the hash
uint32_t h0, h1, h2, h3;

void md5(uint8_t *initial_msg, size_t initial_len) {

    // Message (to prepare)
    uint8_t *msg = NULL;
    int new_len;
    uint32_t bits_len;
    int offset;
    uint32_t *w;
    uint32_t a, b, c, d, i, f, g, temp;

    // Note: All variables are unsigned 32 bit and wrap modulo 2^32 when
calculating

    // r specifies the per-round shift amounts
    const uint32_t r[] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12,
17, 22,
                           5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9,
14, 20,
                           4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11,
16, 23,
                           6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10,
15, 21};
```

```
// Initialize variables - simple count in nibbles:
h0 = 0x67452301;
h1 = 0xefcdab89;
h2 = 0x98badcfe;
h3 = 0x10325476;

// Pre-processing: adding a single 1 bit
//append "1" bit to message
/* Notice: the input bytes are considered as bits strings,
   where the first bit is the most significant bit of the byte.[37] */

// Pre-processing: padding with zeros
//append "0" bit until message length in bit ≡ 448 (mod 512)
//append length mod (2 pow 64) to message

for(new_len = initial_len*8 + 1; new_len%512!=448; new_len++);
new_len /= 8;

    msg = (uint8_t*)calloc(new_len + 64, 1); // also appends "0" bits
                                             // (we alloc also 64 extra bytes...)
    memcpy(msg, initial_msg, initial_len);
    msg[initial_len] = 128; // write the "1" bit

    bits_len = 8*initial_len; // note, we append the len
    memcpy(msg + new_len, &bits_len, 4);           // in bits at the end of
the buffer

    // Process the message in successive 512-bit chunks:
    //for each 512-bit chunk of message:
    for(offset=0; offset<new_len; offset += (512/8)) {

        // break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15
        w = (uint32_t *) (msg + offset);

#ifdef DEBUG
        printf("offset: %d %x\n", offset, offset);

        int j;
        for(j =0; j < 64; j++) printf("%x ", ((uint8_t *) w)[j]);
        puts("");
#endif

        // Initialize hash value for this chunk:
        a = h0;
        b = h1;
        c = h2;
        d = h3;

        // Main loop:
        for(i = 0; i<64; i++) {

            if (i < 16) {
                f = (b & c) | ((~b) & d);
                g = i;
            } else if (i < 32) {
                f = (d & b) | ((~d) & c);
                g = (5*i + 1) % 16;
```

```
        } else if (i < 48) {
              f = b ^ c ^ d;
              g = (3*i + 5) % 16;
        } else {
              f = c ^ (b | (~d));
              g = (7*i) % 16;
        }

         temp = d;
        d = c;
        c = b;
        b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]);
        a = temp;

    }

    // Add this chunk's hash to result so far:
        h0 += a;
        h1 += b;
        h2 += c;
        h3 += d;

    }

    // cleanup
    free(msg);

}

int main(int argc, char **argv) {
    char *msg = argv[1];
    size_t len;
    int i;
    uint8_t *p;

    if (argc < 2) {
        printf("usage: %s 'string'\n", argv[0]);
        return 1;
    }

    len = strlen(msg);

    // benchmark
    for (i = 0; i < 1000000; i++) {
        md5((uint8_t*)msg, len);
    }

    //var char digest[16] := h0 append h1 append h2 append h3 //(Output is in
little-endian)

    // display result

    p=(uint8_t *)&h0;
    printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h0);

    p=(uint8_t *)&h1;
    printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h1);
```

```
    p=(uint8_t *)&h2;
    printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h2);

    p=(uint8_t *)&h3;
    printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h3);
    puts("");

    return 0;
}
```





نمونه خروجی با برنامه پیاده سازی شده

کلمه ارائه شده : mohammadzand

MD5 : 176971922422211156262211324868108525516112718165 13

یک نمونه تبدیل آنلاین :

MD5 Creator

mohammadzand

Create Md5

mohammadzad in MD5 :

994c61035eba4446a5946d0b3216c16c Copy to ClipBoard

mohammadzad in SHA1 :

f82b3a2073e1b0bac0f356ae2c0f12d6c2073f1f Copy to ClipBoard

mohammadzad in md5(sha1) :

8a9a48e22e1eec5c456c8450d48457d7 Copy to ClipBoard

mohammadzad in sha1(md5) :

7012bfbc6fb64772cf8b2dd166842886ce91c667 Copy to ClipBoard

MD5 Creator | What is MD5?

Online Md5 Creator

Webmaster Sucks

MD5 Generator MD5 Cracker Password Generator MD5 Encryption

MD5 Generator

This is a simple tool that will generate a MD5 hash based on a string.

String:

mohammadzand

Generate

MD5 hash:

50ba540651 5b2df2f9b0ec67e4f6c611

Privacy note: We take your privacy very seriously. The encryption process uses Javascript, so your input doesn't even reach our server. We do not store or monitor your input in any way.

What is MD5?

Message-Digest 5, known simply as MD5, is one of the quickest and simplest ways to add security to the files and messages that you send and transfer.

It looks complicated, but it actually relies on a few simple ideas. To get a MD5 hash all you need to do is input your message string into a MD5 generator. This is an application that will apply an algorithm to the string – the MD5 hash.

Once you have it, send the MD5 hash to the recipient and instruct them to put the string into the same MD5 generator. If they get an identical hash, then they got the exact file or message you sent.

Privacy Policy - © 2013 MD5.net

منبع توضیحات :

http://en.wikipedia.org/wiki/MD5

منبع دریافت نمونه پیاده سازی ها :

http://www.codeproject.com
http://www.iranianexperts.ir
http://www.iranianexperts.com
http://groups.yahoo.com/group/IranianExperts

سایر منابع :

http://www.md5.net
http://www.md5-creator.com

سایت شخصی محمد زند :

http://irexperts.ir
http://xand.ir