BITGRAVITY

BitGravity provides a URL signature mechanism for securing access to your files. Access can be restricted on the basis of an expiration date (to implement short-lived URLs) and/or on the basis of the geographic location of the client requesting the download (to restrict the download of certain files to certain countries). Any tampering with the URL in an attempt to circumvent these restrictions will result in an invalid URL.

Overview

BitGravity provides clients with the ability to tokenize URLs linking to the content hosted on the global BitGravity network. The main purpose of this mechanism is to protect client bandwidth by stopping "hot linking" of content through other websites, or to prevent URL leakage. Many clients also use this feature to "lock down" their content by implementing short-lived URLs. This mechanism simply enables web developers to dynamically generate the URLs linking to their content (hosted on the BitGravity network) using server side scripting languages such as JavaScript or PHP. These URLs are no longer static/permanent links to files; rather they are generated behind the authentication safeguards put in place on client websites based on their business intelligence rules. Please bear in mind, this will not prevent the downloading of content on end-user computers. This mechanism enables safeguarding of bandwidth on the BitGravity network and can stop a vast majority of end-users from sharing the URLs. A simple example follows (links are not actual files):

Static non-secured link:

http://www.bitgravity.com/testing/image.jpg

This file will be served no matter where it's being accessed from. End-users can copy+paste this in an email, use it on their own website, or post it on a forum for others to access.

Dynamic secured link:

http://www.bitgravity.com/testing/image.jpg?e=1182665958&a=US&h=886dbef7390dfd70aea27fd41e4

This link has been secured using an MD5 hash code plus an expiry time-stamp and a geolocation restriction. This file will only be served 30 seconds after the link was generated (when the page initially loaded, using PHP), will only be viewable within the United States, and can not be tampered with. If someone passed this link around or modified anything in the URL sequence, clicking on it would result in a "403 - Forbidden" error code.

Additional notes:
BitGravity URLs can be secured based on the following criteria:

- An expiry time
- An optional list of allowed countries
- An optional list of disallowed countries
- An optional list of allowed metros
- An optional list of disallowed metros
- An optional IP address
- An optional user agent match string

Implementing Secure URLs

Having URLs expire shortly after they are sent to the user provides an easy way to protect content from casual sharing of links in email or the posting of links on bulletin boards.

There are a few aspects of this that may not be obvious. Firstly, the expiry time can be very short, even just a few seconds. The URL only has to be valid for the start of the request not the duration of the download. We recommend your download links be a server-side script on your own server that validates the user, generates a secure URL with a few seconds until expiration (say 10) and then redirects the user's browser to that secure URL. This script can also implement whatever sort of "one time" rules your application wants to enforce on downloading content. The key to using short expiry times is that the secure URLs must be generated "on the fly" in response to the user clicking on a download link (or a play button on a player widget) rather than being generated as part of a web page that can sit on a user's screen for an unknown length of time. Note that if you are using progressive downloads and are scrubbing through the file (using the "start" parameter) you'll need to generate a new secure URL each time since that parameter is included in the secure hash to prevent its misuse (however, see the discussion of "apstart" below).

While it is possible to generate secure URLs in a Flash client on the end user's computer we don't recommend this because it means the shared key has to be downloaded to every end user (extracting it from a compiled Flash application is non-trivial but certainly possible). There are other issues too, for example you'll need to come up with a mechanism for getting the correct time from an NTP synchronized server since the time on the client PC can't be relied upon to be accurate to within minutes, never mind seconds. The only practical limit on how short an expiry time can be used when generating the URL on the client is the time it takes for the client to do a DNS lookup of the BitGravity hostname (this can be mitigated by making sure the hostname has already been accessed before the URL is generated).

If expiration isn't the preferred method of controlling the reuse of download URLs then you might consider using a zero expiry time (which means "don't expire") and encoding the user's IP address in the URL to limit downloads to that client. Again a server-side script, which generates a redirect to the secure URL is the recommended way of implementing this. Some end-user networking arrangements make this approach problematic however (e.g. devices that aggregate multiple network connections) and we don't recommend the use of IP addresses in SecureAccess URLs for all situations.

URL Signatures

URLs are protected from alteration by generating a MD5 digest of the valid URL and checking that against the actual URL requested. Thus, if an end user tries to alter the expiry time or allowed country for the URL the server will reject the request since the MD5 is no longer valid. A **shared secret** is used to prevent unauthorized regeneration of the MD5 (the "secret" is a password known to BitGravity and you, the owner of the protected content).
Secure URLs can be generated using the provided PHP script (see **bg_gen_secure_uri** at the end of this document). Secure URLs are typically generated on the web server that serves the pages with links to the content. Any language that provides the means to generate an MD5 digest of a string can be used to implement the simple algorithm, which generates the secure URL (the PHP script can be used as pseudo code to do this).

URI Signature Algorithm

A secure URI consists of several parts:

- The file to be served (e.g. /acmecompany/content/secure.ext)
- An expiry time (e.g. e=1182665958). This is not optional but can be "0" to indicate "does not expire".
- An optional list of allowed countries (e.g. a=US,CA)
- An optional list of disallowed countries (e.g. d=LY,CD)
- An optional list of allowed metro codes (e.g. am=807,828)
- An optional list of disallowed metro codes (e.g. dm=609)
- An optional IP address (e.g. i=12.34.56.78)
- An optional user agent match string (e.g. u=Firefox)
- Optional progressive streaming parameters (e.g. start=0 or end=2345678)

- The MD5 hash that secures the URI (e.g. h=886dbef7390dfd70aea27fd41e459e7f)

The expiration time is specified as a standard POSIX timestamp (seconds since January 1 1970 00:00:00 UTC). The server compares this timestamp with the current time to determine if the URL has expired. Allowing or disallowing metro codes provides a way to limit distribution of content within the US by market. For example, the metro code for San Francisco-Oakland-San Jose CA is 807 and specifying "am=807" means that the content can only be accessed within that area. More commonly, specifying a disallowed metro (or metros) allows a "blackout" to be enforced for the content (e.g. a sporting event that is not available online in the market where the live event is taking place). Metro codes are used by the advertising industry to divide the US into markets. A convenient list is available here:

http://www.google.com/apis/adwords/developer/adwords_api_us_metros.html

A signature (MD5 hash) is generated for the URL using the following data:

> *"secret" + "file" + "?e=timestamp" + "&a=allowed-countries" + "&d=disallowed-countries" + "&am=allowed-metros" + "&dm=disallowed-metros" + "&i=allowed-ip" + "&u=allowed-useragent" + "&start=progressive-start" + "&end=progressive-end"*

The timestamp can be 0 (zero) which disables expiry of the URL (i.e. it never expires). Allowed and disallowed parameters are mutually exclusive (i.e. it is only valid to specify one or none of them).

For example, with:

- a file of **/acmecompany/content/protected.flv**
- with a shared secret of **mySecret**
- an allowed country of **US**
- and an expiry timestamp of **1182665958**

the resulting hash, MD5(**mySecret/acmecompany/content/protected.flv?e=1182665958&a=US)** will be **ec41f550878f45d9724776761d6ac416.**

The resulting URL for this secure download would be:
http://bitcast-g.bitgravity.com/
acmecompany/content/protected.flv?e=1182665958&a=US&h=ec41f550878f45d9724776761d6ac416

SecureAccess and Advanced Progressive

SecureAccess can be used with BG Advanced Progressive to lock down the URLs used to retrieve streaming content. If start or end parameters are used to control the retrieval of FLV content they should be included in the hash. In this way the end user can't manipulate the portions of the content retrieved by modifying the URL.

There are a number of additional parameters to be used in conjunction with BG Advanced Progressive. These parameters shall not be included with the MD5 hash generation to avoid the need for regeneration of a new hash with each Advanced Progressive request.

- apstart
- apend
- starttime
- endtime

Summary of SecureAccess + Advanced Progressive parameters

Parameter	Hashed	Usage
start	Yes	Restrict start byte offset within an AP enabled FLV file.
end	Yes	Restrict end byte offset within an AP enabled FLV file.
apstart	No	Allow scrubbing using start byte offset within an AP enabled FLV file.
apend	No	Allow scrubbing using end byte offset within an AP enabled FLV file.
starttime	No	Allow scrubbing using start time within an AP enabled FLV/MP4 file.
endtime	No	Allow scrubbing using end time within an AP enabled FLV/MP4 file.

In order to use the "apstart", "apend", "starttime", and "endtime" parameters a valid secure URI is required, i.e. the expiry time on the link shall be long enough to allow for scrubbing of video using the Advanced Progressive parameters.

Any other CGI parameters used by the client application should not be included in the hash, only those mentioned above.

PHP Sample Code for Generating Keys

An inline call to PHP's MD5 function can be utilized to generate secure URIs.

```
<?php
$expiry_time = time() + 3600; // 3600 is in seconds. Adjust as desired.
$allowed_countries = "US,CA";
echo "Secure URL:
     <a href=\"http://bitcast-g.bitgravity.com/account_id/secure/file.txt?e=".$expiry_time.
     "&a=".$allowed_countries.
     "&h=".MD5("myPassXYZ/account_id/secure/file.txt?e=".$expiry_time."&a=".$allowed_countries).
     "\">Link</a>";
  ?>
```

The above example generates a secured link to a file that will expire 3600 seconds after page load (or after the time at which the code is executed). This link is only available to visitors from USA and Canada.
Note: the filename passed to the MD5 function shall be the exact BitGravity file path and should not include anything before the BitGravity account ID (i.e. should be specified as: /account_id/directory/file.ext Note the included leading slash). This example assumes the shared secret is set to 'myPassXYZ'.

This PHP script implements generation of secure URIs.

```
<?php

/*
* Generate a BitGravity secure download link for the given parameters.
*
* Copyright (c) 2007-2009 BitGravity LLC.
* All rights reserved.
*/
/*
* Compute a secure URI.
*
* INPUTS:
* $file - base URI, i.e everything after bitgravity.com in file path, including the leading slash
*         (ex. http://bitcast-g.bitgravity.com/acmecompany/content/file.txt)
* $secret - shared secret
* $expiry - expiry in seconds since current time. Use 0 for no expiry
* $allowed_countries - list of allowed countries for GeoIP security (comma separated)
* $disallowed_countries - list of disallowed countries for GeoIP security (comma separated)
* $allowed_metros - list of allowed metros for GeoIP security (comma separated)
* $disallowed_metros - list of allowed metros for GeoIP security (comma separated)
* $allow_ip - restrict URL to this IP address
* $allowed_useragent - restrict URL to requests whose user agent match this regex
* $progressive_start - byte offset indicating the staring point of FLV playback
* $progressive_end - byte offset indicating the ending point of FLV playback
* $extra_params - custom CGI parameters defined by BitGravity customers


IMPORTANT NOTE: this function will return a hashed URL. If you are hashing a video
URL for use in the BitGravity Player embed code, you must replace all instances of
'&' with %26.

eg. echo str_replace("&", "%26", bg_gen_secure_uri('/dean/test2.flv', 'mysecret', 5));

*/
```

```
function bg_gen_secure_uri($file, $secret, $expiry = 0, $allowed_countries = '',
                           $disallowed_countries = '', $allowed_ip = '', $allowed_useragent = '',
                           $allowed_metros = '', $disallowed_metros = '',
                           $progressive_start = '', $progressive_end = '',
                           $extra_params = '') {


      if ($file==''||$secret=='') {
            return false;
      }


      // Construct the values for the MD5 salt ...
      if (substr($expiry,0,1)=='=') {
            $timestamp=substr($expiry,1);
      } else if ($expiry > 0) {
            $now=time();// use UTC time since the server does
            $timestamp=$now+$expiry;
      } else {
            $timestamp=0;
      }

      if ($allowed_countries) {
            $allowed_countries='&a='.$allowed_countries;
      }
      if ($disallowed_countries) {
            $disallowed_countries='&d='.$disallowed_countries;
      }
      if ($allowed_ip) {
            $allowed_ip='&i='.$allowed_ip;
      }
      if ($allowed_useragent) {
            $allowed_useragent='&u='.$allowed_useragent;
      }
      if ($progressive_start!='') {
            $progressive_start='&start='.$progressive_start;
      }
      if ($progressive_end) {
            $progressive_end='&end='.$progressive_end;
      }
      if ($allowed_metros) {
            $allowed_metros='&am='.$allowed_metros;
      }
      if ($disallowed_metros) {
            $disallowed_metros='&dm='.$disallowed_metros;
      }
      if ($extra_params) {
            $extra_params=urldecode($extra_params);
      }

      // Generate the MD5 salt ...
      $salt = $secret . $file . '?e=' . $timestamp . $allowed_countries .
      $disallowed_countries . $allowed_metros . $disallowed_metros . $allowed_ip .
      $allowed_useragent . $progressive_start . $progressive_end;

      // Generate the MD5 hash ...
      $hash_code = md5($salt);
```

```
	// Generate the link ...
	$url = $file . '?e=' . $timestamp . $allowed_countries . $disallowed_countries .
	$allowed_metros . $disallowed_metros . $allowed_ip . $allowed_useragent .
	$progressive_start .   $progressive_end .'&h=' . $hash_code . $extra_params;

	return $url;
}
```


700 Airport Blvd. Burlingame, CA 94010 | Direct 650.262.0004 | Fax 650.353.9915 | www.bitgravity.com

Restricting Content by Geographic Location

URLs can be geographically restricted in the following ways:

- The **allowed** countries can be specified
- The **disallowed** countries can be specified

Countries are specified using the two character ISO 3166 (*http://www.maxmind.com/app/iso3166*) country codes. For example, US specifies the United States and IE specifies Ireland.

Only one of **allowed or disallowed** parameters should be specified. The allowed and disallowed lists can contain a comma-separated list of any number of country codes.

In the past an "unlock" parameter could be used in conjunction with server-side geographic restrictions. This is no longer an available option (better functionality is available by generating URLs with appropriate allowed or disallowed country lists).

Error Reporting

The server either successfully returns the requested content or it indicates that access was not allowed due to either expiration or country of access using HTTP code 403 ("Forbidden"). If the URL is invalid (i.e. has been altered since it was generated) then HTTP code 400 ("Bad request") is returned to the client.