Private Blobs RFC for customer feedback

Abstract

This RFC outlines the implementation of private blobs in Vercel Blob Storage. Private blobs provide secure, controlled access to stored content without public URLs, allowing customers to manage access to their stored data.

Background

By default, blobs uploaded to Vercel Blob Storage are accessible via public URLs. While these URLs include unguessable components, many use cases require strictly private access with controlled distribution.

Common use cases for private blobs include:

- Temporary digital downloads (e.g., purchased ebooks)
- User-generated or user-uploaded content with restricted access
- Media file storage with controlled access
- Any website willing to control who has access to which blobs

Private Blob Implementation

1. Uploading Private Blobs

```
import { put } from '@vercel/blob';

const blob = await put('/user123/document.pdf', content, { ac
cess: 'private' });

console.log(blob);
// {
//   pathname: '/user123/document.pdf',
```

```
//   contentType: 'application/pdf',
//   contentDisposition: 'attachment; filename="document.pd
f"'
// }
```

Note: Private blobs don't generate public URLs and don't add random suffixes to pathnames.

2. Accessing Private Blobs

There are three methods for accessing private blobs:

Method 1: Temporary URLs

Generate time-limited URLs for blob access, similar to S3 pre-signed URLs:

```
import { createPrivateBlobUrl } from '@vercel/blob';

const privateBlobUrl = createPrivateBlobUrl(
  '/user123/document.pdf',
  { expiresAt: new Date(Date.now() + 3600000) } // 1 hour exp
iry
);

// URL can be shared with authorized users
```

For directory-level access:

```
import { createPrivateBlobBaseUrl } from '@vercel/blob';

const privateBlobBaseUrl = createPrivateBlobBaseUrl(
  '/user123/',
  { expiresAt: new Date(Date.now() + 3600000) }
);

const fileUrl = `${privateBlobBaseUrl}document.pdf`;
```

Temporary URLs are validated and served directly by the Vercel Edge Network.

Method 2: Blob Token Authentication

Stream blobs through your application's serverless functions using the Vercel Blob SDK:

```
import { get } from '@vercel/blob';

export default async function handler(req, res) {
  const { stream, blob } = await get('/user123/document.pd
f');

  return new Response(stream, {
    headers: {
      'content-type': blob.contentType
    }
  });
}
```

This method provides maximum control over blob access through your application's authentication and authorization logic.

Method 3: Edge Middleware Authorization

Authorize blob access directly at the edge using Vercel Edge Middleware:

```
// middleware.ts
import { NextResponse } from 'next/server';
import { createBlobAccess } from '@vercel/blob';

export async function middleware(request) {
  if (request.nextUrl.pathname.startsWith('/api/blobs/')) {
    const user = await auth(request);

    if (await isAuthorized(user, request.nextUrl.pathname)) {
      const response = NextResponse.next();
```

```
      response.headers.set('x-blob-access', createBlobAccess
(request.nextUrl.pathname));
      return response;
    }

    return new NextResponse(null, { status: 403 });
  }

  return NextResponse.next();
}
```

This method combines edge performance with authorization capabilities, allowing you to validate user permissions before serving private blobs.