Fork me on GitHub

Encryption in the Cloud

Martin Vasko

CEO and Founder of

expressFlow™

Code for this talk available on GitHub:
https://github.com/martinvasko/expressFlow

Email: martin@expressflow.com
Twitter: @expressflow
Facebook: fb.com/encrypt.files.with.a.drop

The outline

1. Drag-and-Drop into the Google Cloud

2. SSL and Google App Engine

3. Encrypt your files in the Google Cloud

4. Manage the keys – the tricky part

5. Conclusion and outlook

The outline

1. **Drag-and-Drop into the Google Cloud**

2. SSL and Google App Engine

3. Encrypt your files in the Google Cloud

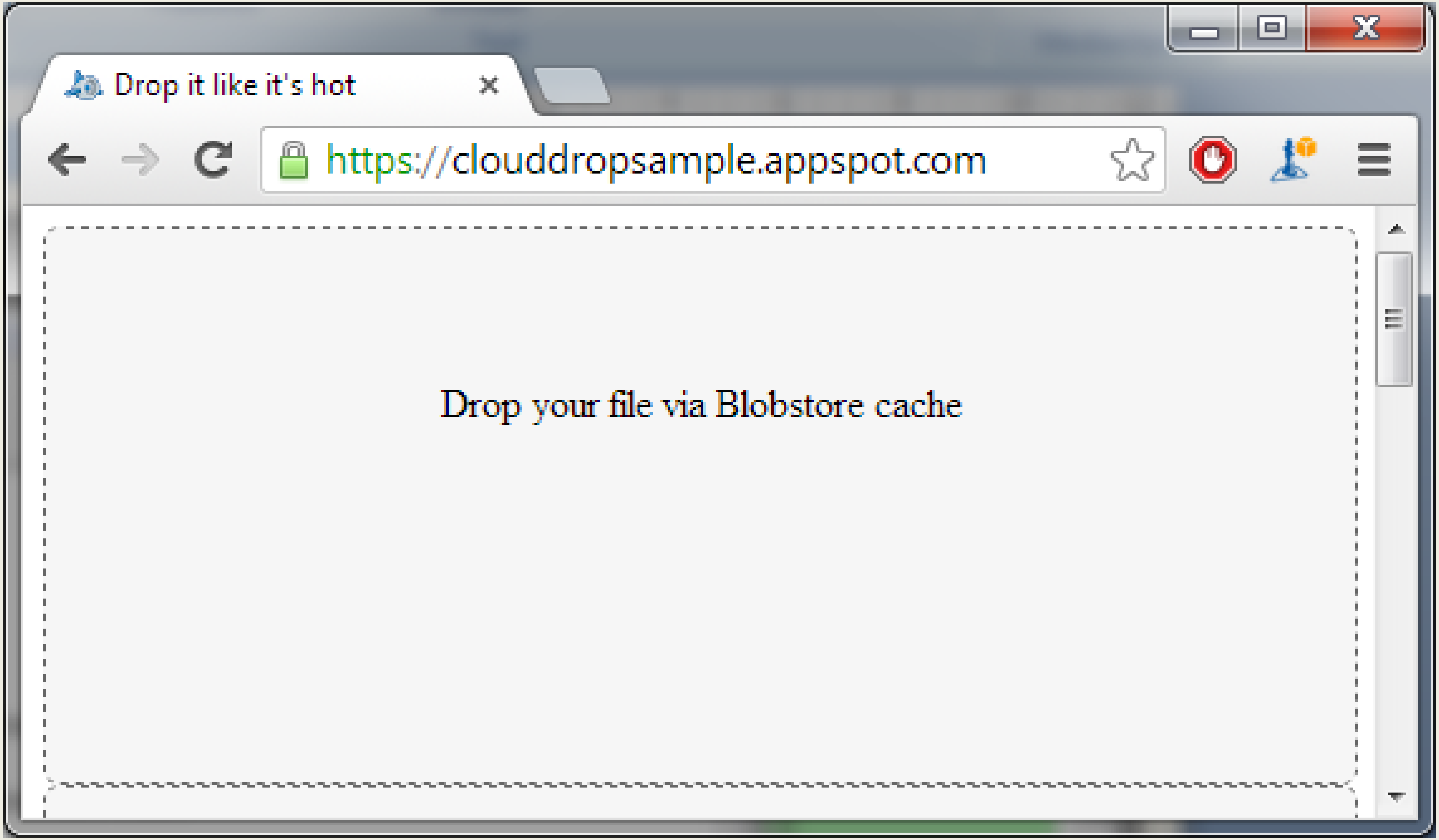
4. Manage the keys – the tricky part

5. Conclusion and outlook

Get files into the Google Cloud

- Simplicity is key!
- Focus of our example lies on Browser/Desktop scenarios
- Requirements:
  - Basic HTML5 know how (mostly File W3C)
  - Google App Engine Data store know how (Blobstore vs. Google Cloud Storage)

Our goal for the next 15 min



Drop your file via Blobstore cache

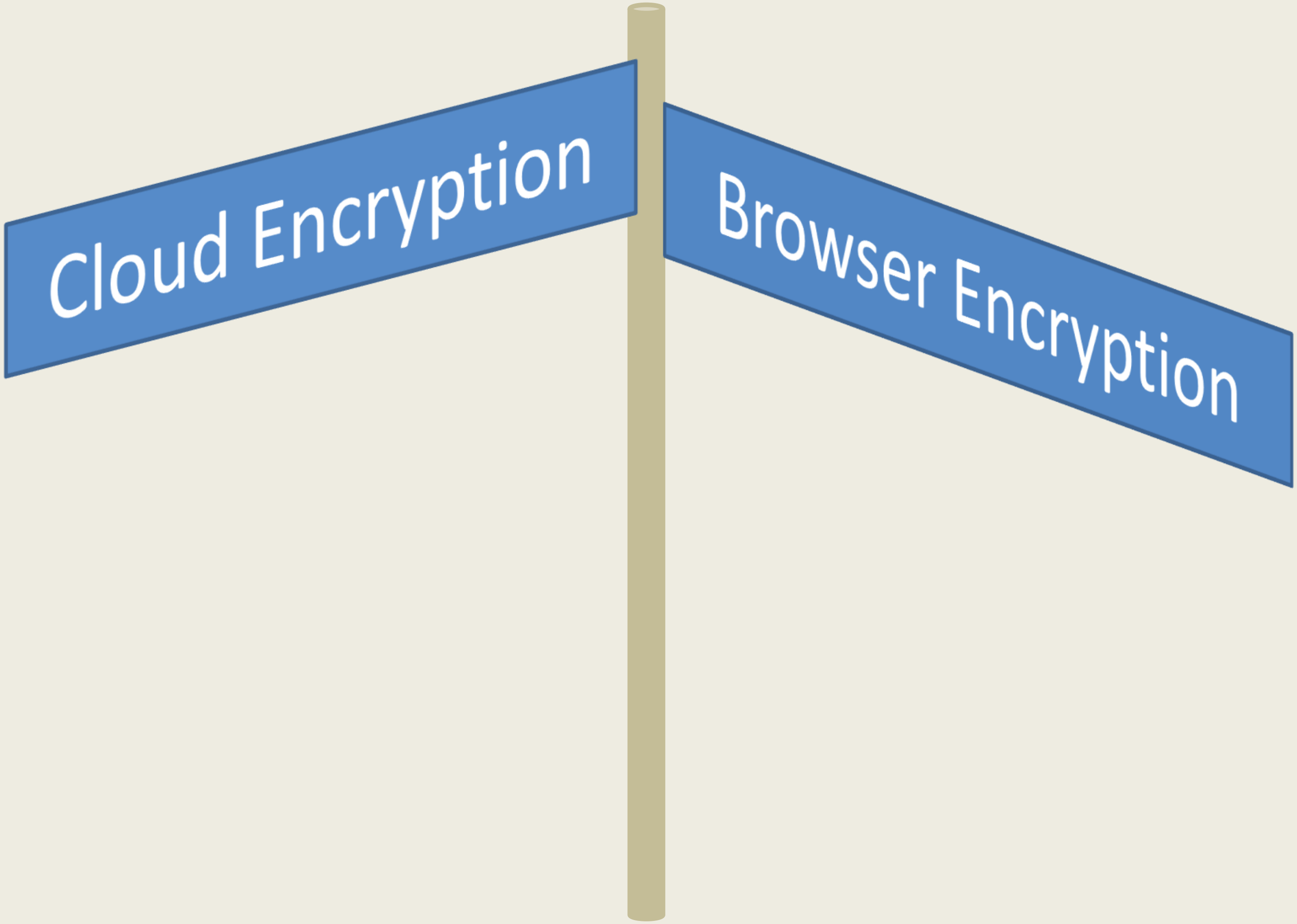
The ingredients

- HTML 5
  - JQuery (we use jquery-filedrop)
- Google App Engine (Java)
  - Servlets
  - Blobstore
  - Google Cloud Storage client
- Google Cloud Storage
  - Buckets / objects

Real world scenario

- HTML 5
  - jQuery File Upload
  - Dropzone.js
  - many more – use them!
- Google App Engine frameworks
  - Spring (and how to optimize it)
  - Objectify (JDA API for GAE Datastore)
  - many more – use them!

Different directions ahead

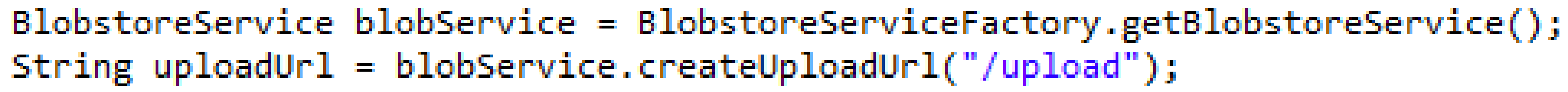


Local vs. Cloud Encryption

	Local (Browser-based) encryption	Cloud Encryption
Security	High	Medium
Complexity	High	Medium
Portability	Medium	High
Integration	Low	High

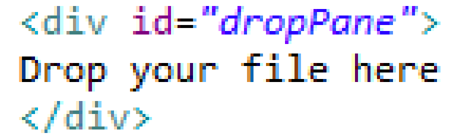
The scaffold for the client …

- An upload-url (obviously)



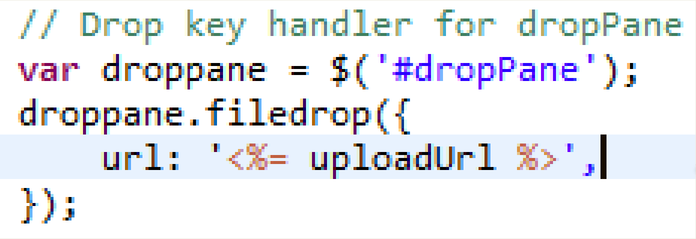
```
BlobstoreService blobService = BlobstoreServiceFactory.getBlobstoreService();
String uploadUrl = blobService.createUploadUrl("/upload");
```

- A <div> to drop on



```
<div id="dropPane">
Drop your file here
</div>
```

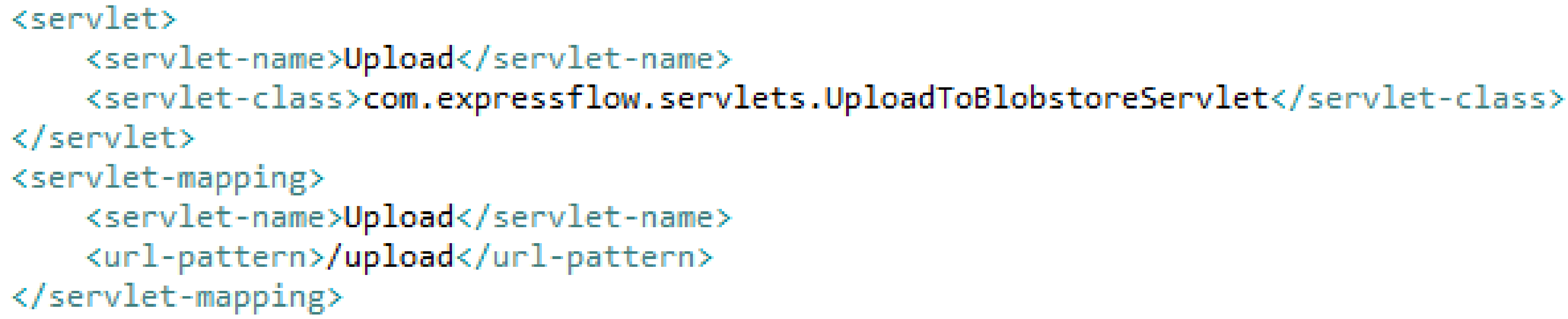
- JQuery handlers for the filedrop event



```
// Drop key handler for dropPane
var droppane = $('#dropPane');
droppane.filedrop({
    url: '<%= uploadUrl %>',
});
```

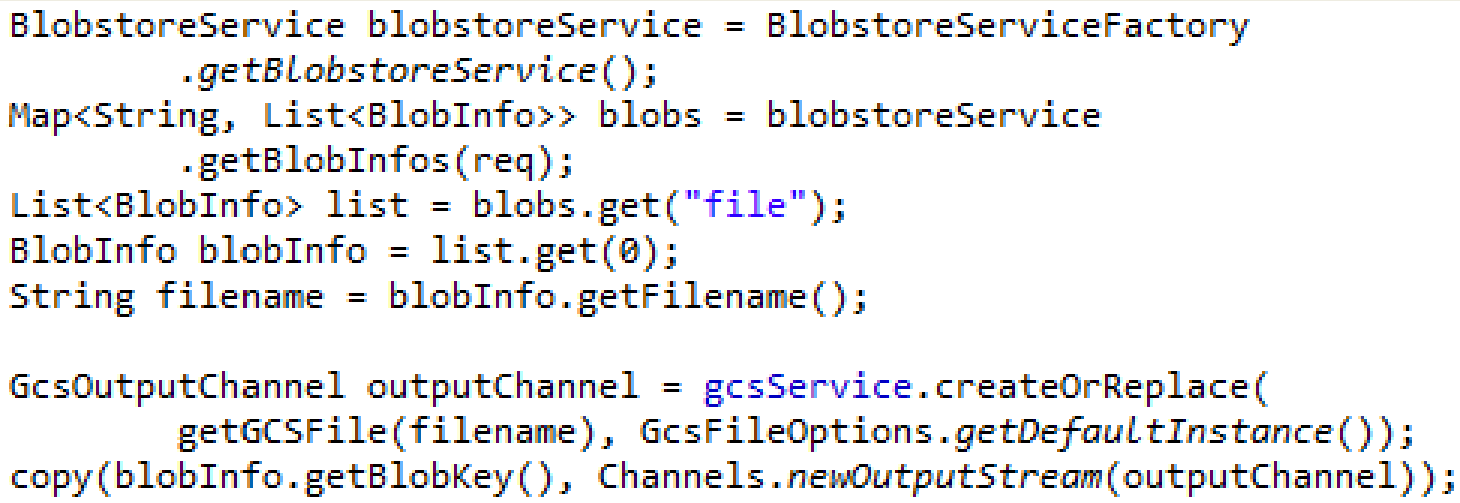
… and for the server

- web.xml



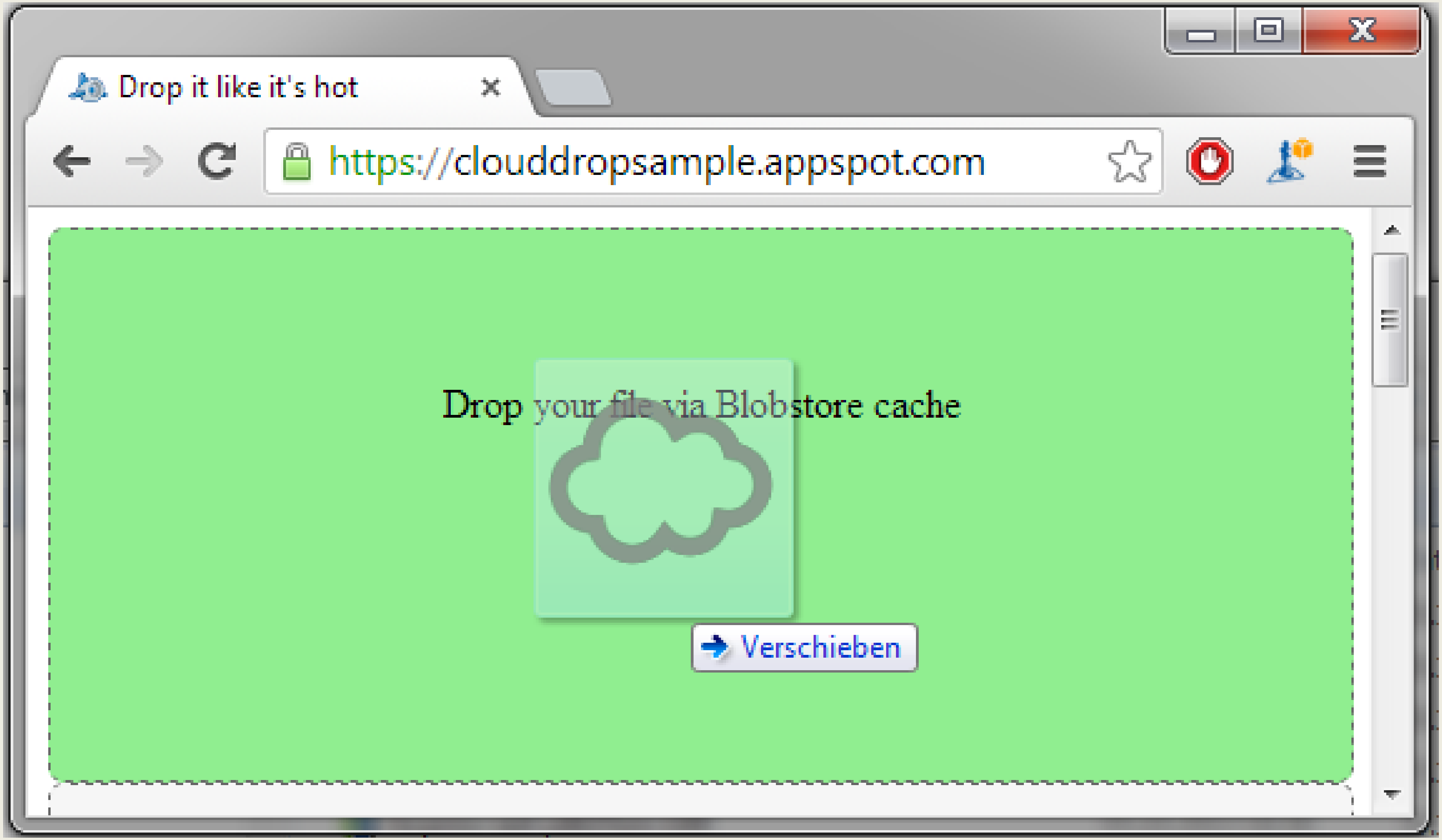
```
<servlet>
    <servlet-name>Upload</servlet-name>
    <servlet-class>com.expressflow.servlets.UploadToBlobstoreServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Upload</servlet-name>
    <url-pattern>/upload</url-pattern>
</servlet-mapping>
```

- HTTP POSTs to [/upload]are handled by



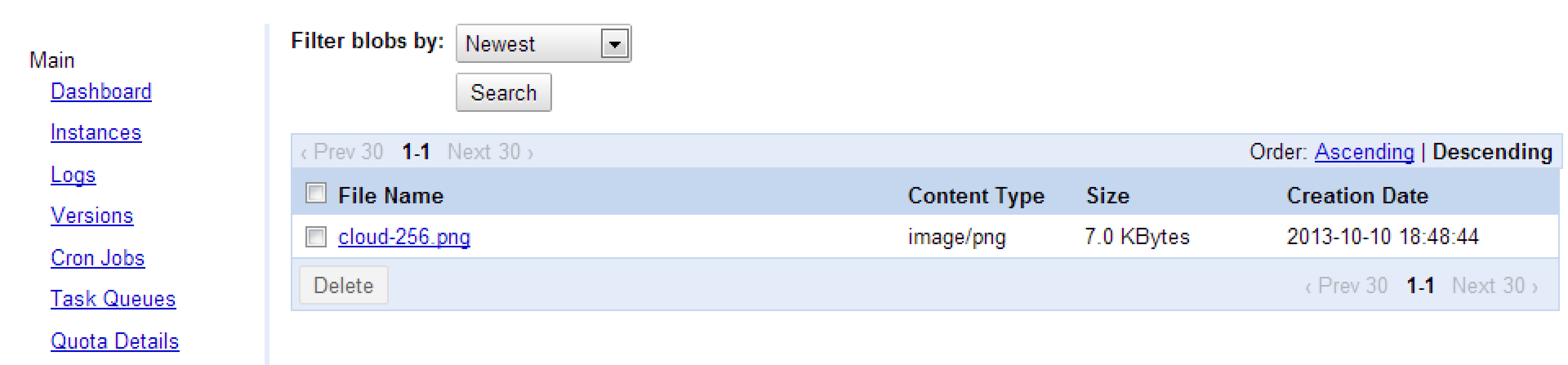
```
BlobstoreService blobstoreService = BlobstoreServiceFactory
        .getBlobstoreService();
Map<String, List<BlobInfo>> blobs = blobstoreService
        .getBlobInfos(req);
List<BlobInfo> list = blobs.get("file");
BlobInfo blobInfo = list.get(0);
String filename = blobInfo.getFilename();

GcsOutputChannel outputChannel = gcsService.createOrReplace(
        getGCSFile(filename), GcsFileOptions.getDefaultInstance());
copy(blobInfo.getBlobKey(), Channels.newOutputStream(outputChannel));
```

Dropping a file onto the website…



Drop your file via Blobstore cache

..stores it in the Blobstore



Main
- Dashboard
- Instances
- Logs
- Versions
- Cron Jobs
- Task Queues
- Quota Details

Filter blobs by: Newest

‹ Prev 30 **1-1** Next 30 › Order: Ascending | **Descending**

□ File Name	Content Type	Size	Creation Date
□ cloud-256.png	image/png	7.0 KBytes	2013-10-10 18:48:44

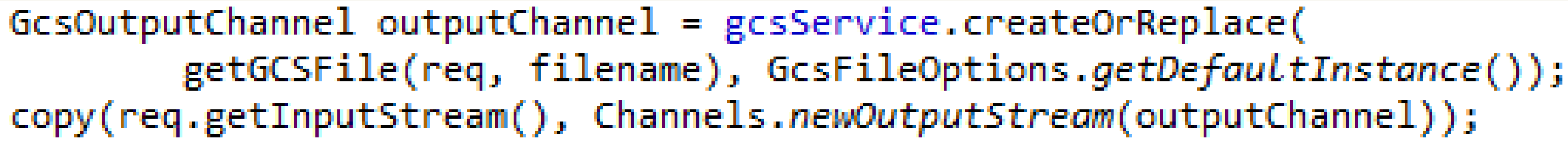
‹ Prev 30 **1-1** Next 30 ›

Nice!

But what about the Google Cloud Storage?

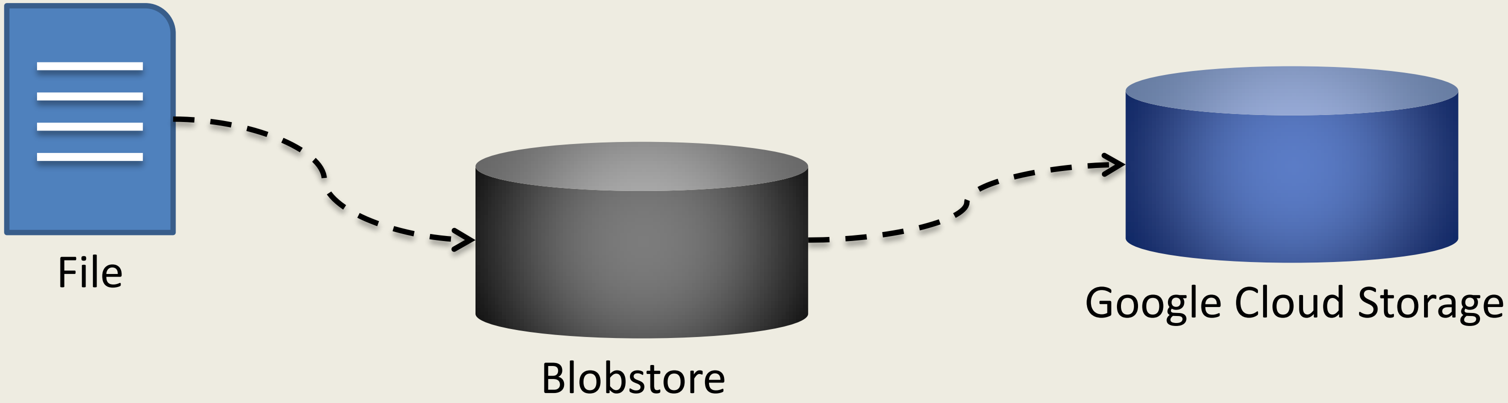
It’s there too! Remember?

Server side copy to Cloud Storage



```
GcsOutputChannel outputChannel = gcsService.createOrReplace(
        getGCSFile(req, filename), GcsFileOptions.getDefaultInstance());
copy(req.getInputStream(), Channels.newOutputStream(outputChannel));
```

The uploaded file traversal in detail:

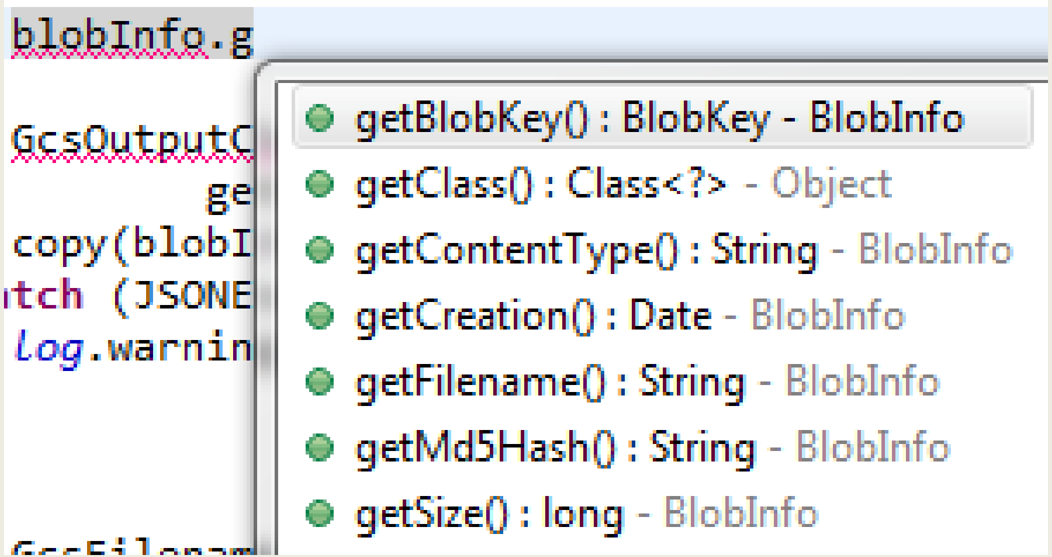


Why the interlude at `Blobstore`?

Let’s take a closer look.

Upload to Blobstore

- Handy BlobInfo

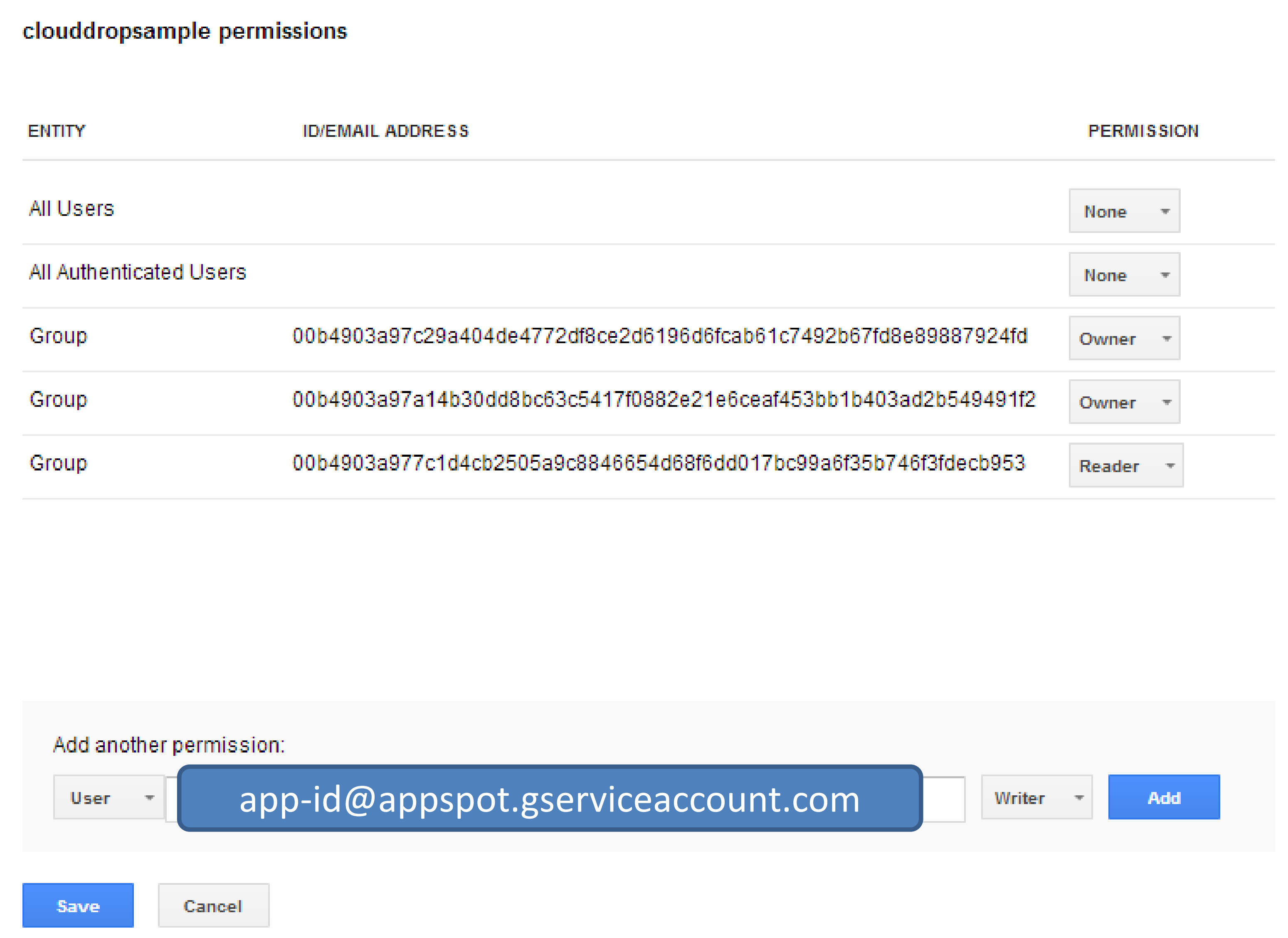


```
blobInfo.g

GcsOutputC
         ge
copy(blobI
tch (JSONE
log.warnin
```

- getBlobKey() : BlobKey - BlobInfo
- getClass() : Class<?> - Object
- getContentType() : String - BlobInfo
- getCreation() : Date - BlobInfo
- getFilename() : String - BlobInfo
- getMd5Hash() : String - BlobInfo
- getSize() : long - BlobInfo

- GAE services integrate Blobstore (f.e. ImageService)
- It’s all about the metadata!

Google Cloud Storage Setup



clouddropsample permissions

ENTITY	ID/EMAIL ADDRESS	PERMISSION
All Users		None
All Authenticated Users		None
Group	00b4903a97c29a404de4772df8ce2d6196d6fcab61c7492b67fd8e89887924fd	Owner
Group	00b4903a97a14b30dd8bc63c5417f0882e21e6ceaf453bb1b403ad2b549491f2	Owner
Group	00b4903a977c1d4cb2505a9c8846654d68f6dd017bc99a6f35b746f3fdecb953	Reader

Add another permission:

User | app-id@appspot.gserviceaccount.com | Writer | Add

Where are we now?

1. Drag-and-Drop into the Google Cloud ✓

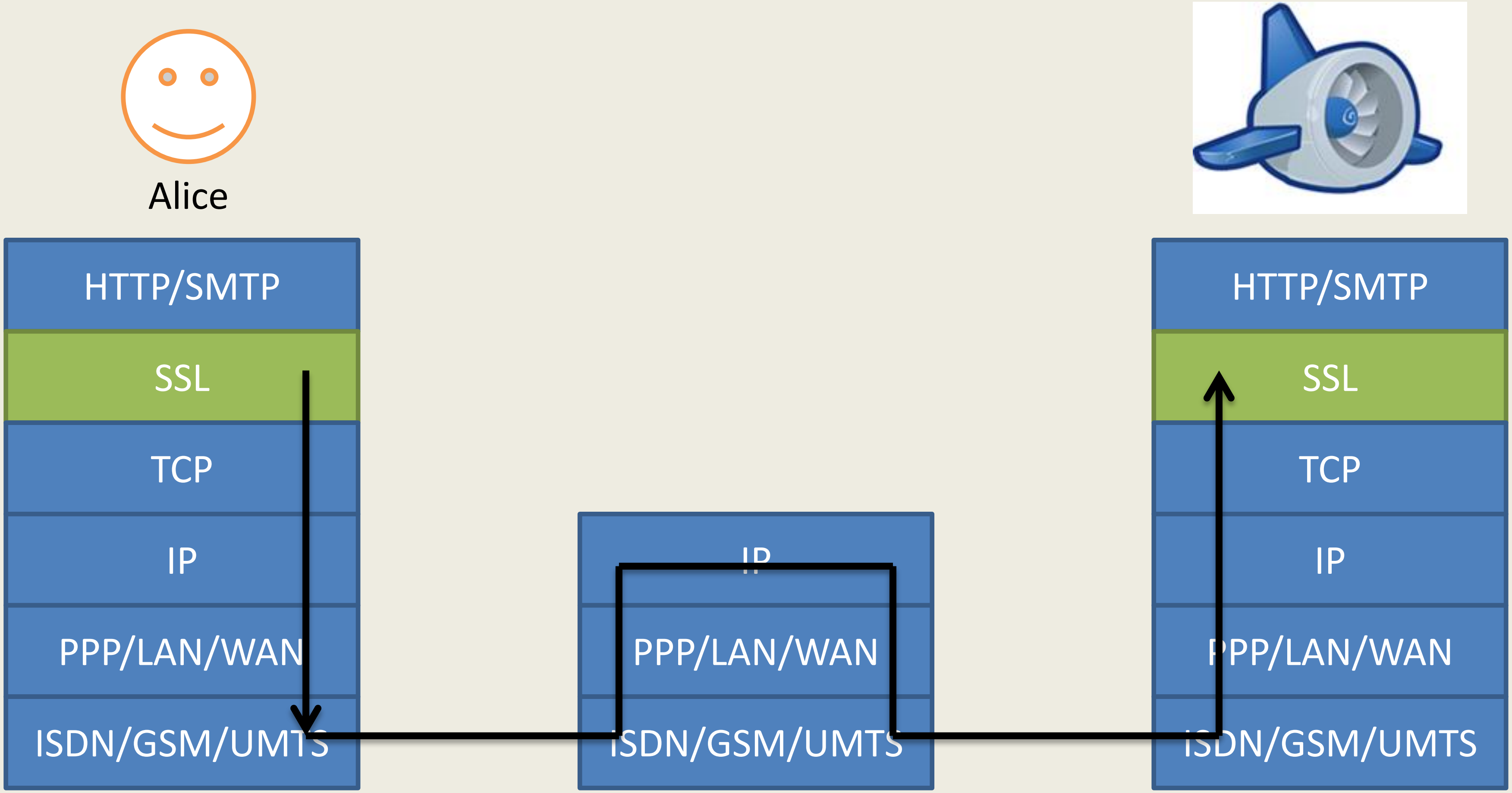
2. SSL and Google App Engine

3. Encrypt your files in the Google Cloud

4. Manage the keys – the tricky part

5. Conclusion and outlook

SSL and Google App Engine



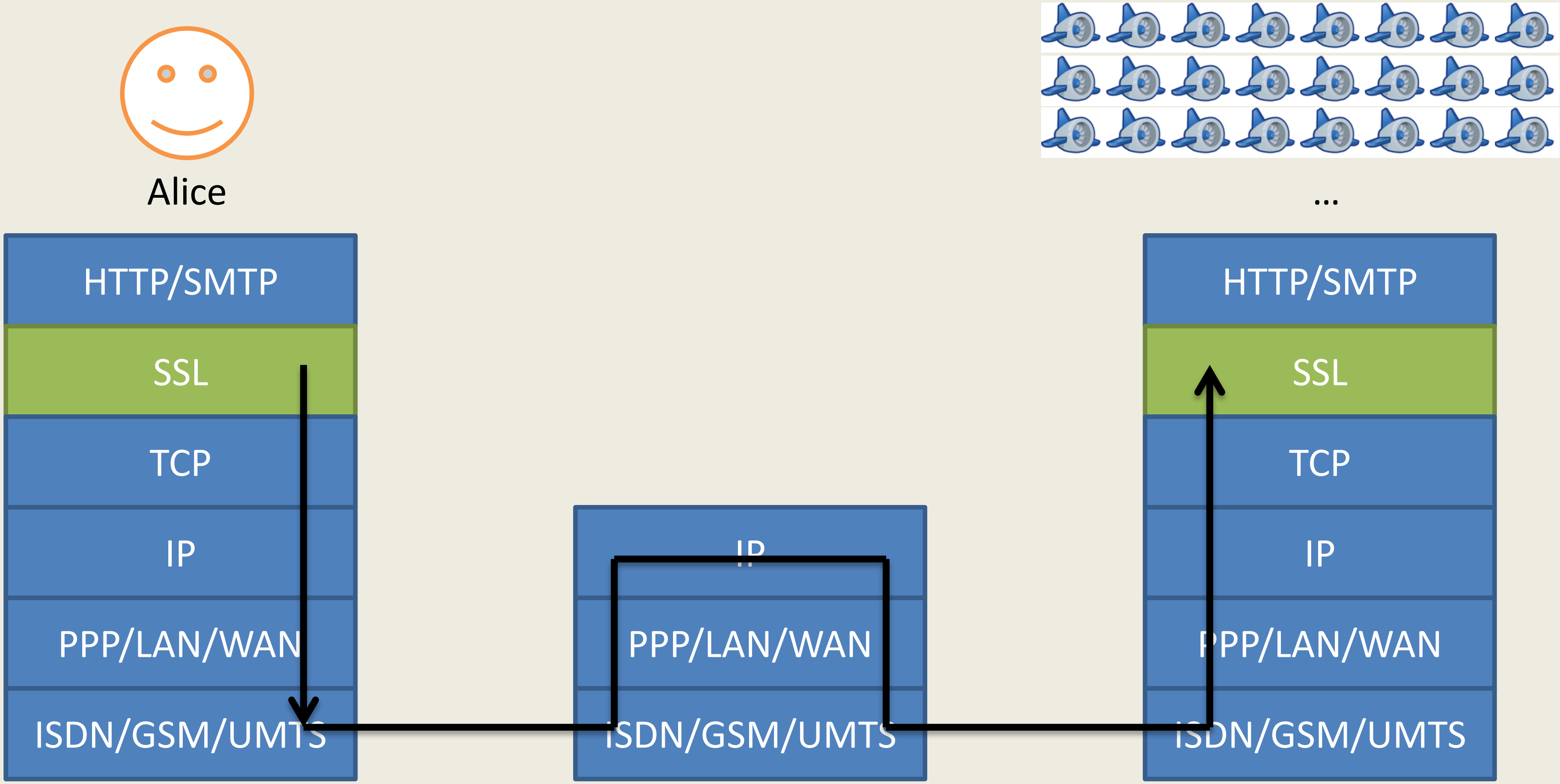
Alice

HTTP/SMTP
SSL
TCP
IP
PPP/LAN/WAN
ISDN/GSM/UMTS

IP
PPP/LAN/WAN
ISDN/GSM/UMTS

HTTP/SMTP
SSL
TCP
IP
PPP/LAN/WAN
ISDN/GSM/UMTS

Actually, this is the case



Alice

…

HTTP/SMTP

SSL

TCP

IP

PPP/LAN/WAN

ISDN/GSM/UMTS

IP

PPP/LAN/WAN

ISDN/GSM/UMTS

HTTP/SMTP

SSL

TCP

IP

PPP/LAN/WAN

ISDN/GSM/UMTS

Securing endpoints in the cloud…

- … is tricky as it is an N-to-N encryption

Google provides two solutions for SSL:

- Server Name Indication (SNI)

  Multiple domains share one IP address

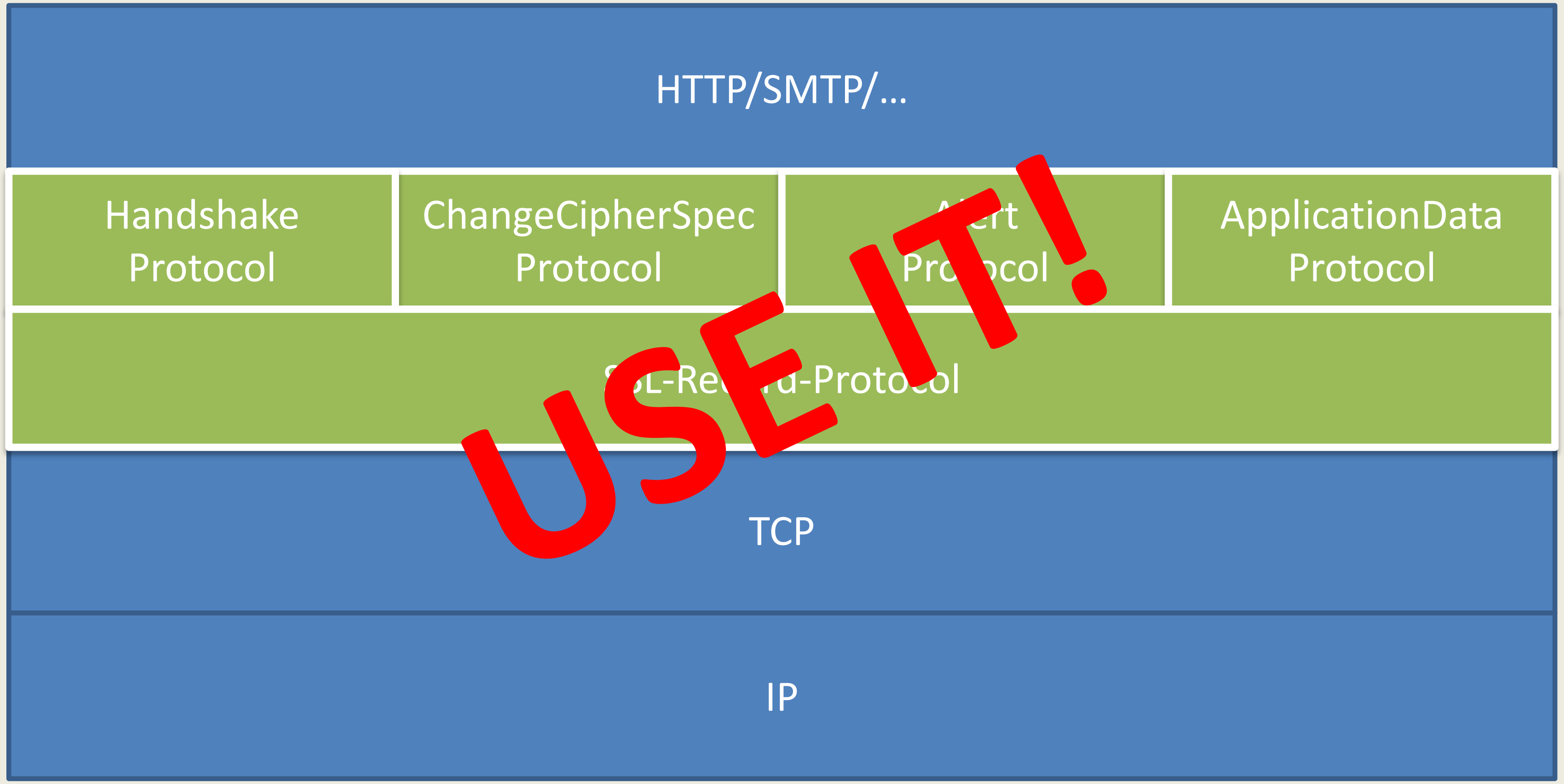
Pros	Cons
Cheap, easy setup	Not supported by old browsers (like Android-Browsers > Honeycomb)

- Virtual IP

  Dedicated IP address assigned to your app

Pros	Cons
Supported by all browsers that support SSL	(Very) Expensive

SSL for data in transit



HTTP/SMTP/…

Handshake Protocol

ChangeCipherSpec Protocol

Alert Protocol

ApplicationData Protocol

SSL-Record-Protocol

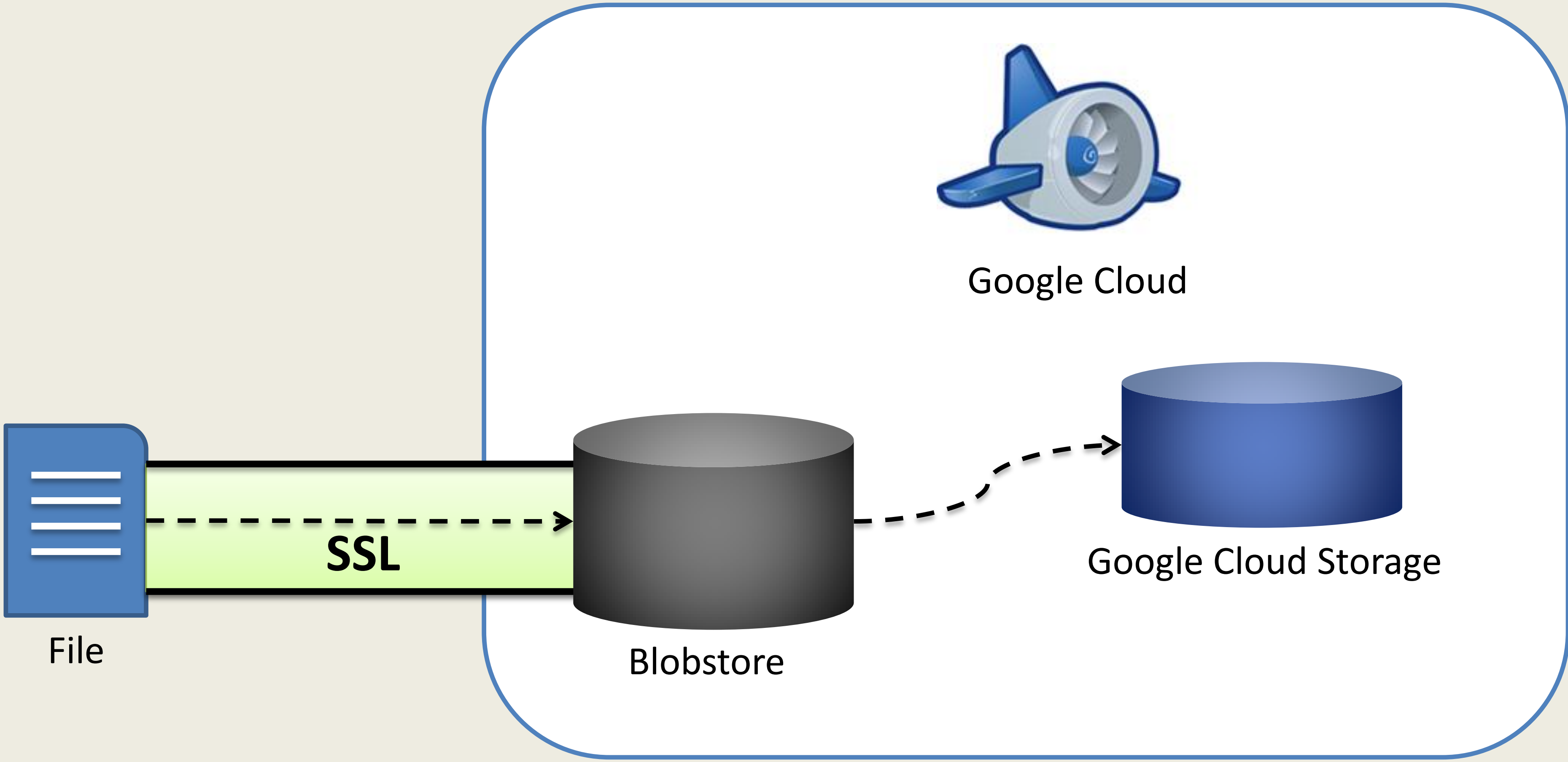
TCP

IP

USE IT!

…

SSL in our example



Where are we now?

1. Drag-and-Drop into the Google Cloud ✓

2. SSL and Google App Engine ✓

3. Encrypt your files in the Google Cloud

4. Manage the keys – the tricky part

5. Conclusion and outlook

Data Encryption – Rocket science?

OFB Mode

Initialization vector

Data padding

Key size

Cipher

Block size

Cipher text

Key Generator

Zero Padding

ECB Mode

PKCS5/7 Padding

PCBC Mode

Plain text

XOR

CFB Mode

CBC Mode

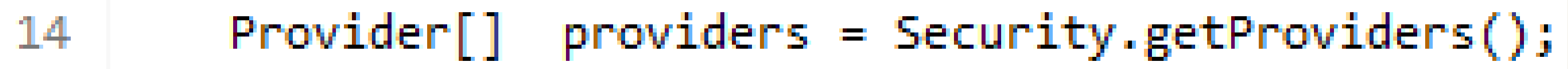
Well, it's tricky

- … but NO Rocket Science!
- What do we need?
  - Cryptography Architecture
  - File Access
  - Performance and Scalability
- What do we have already?
  - ✓ File Access

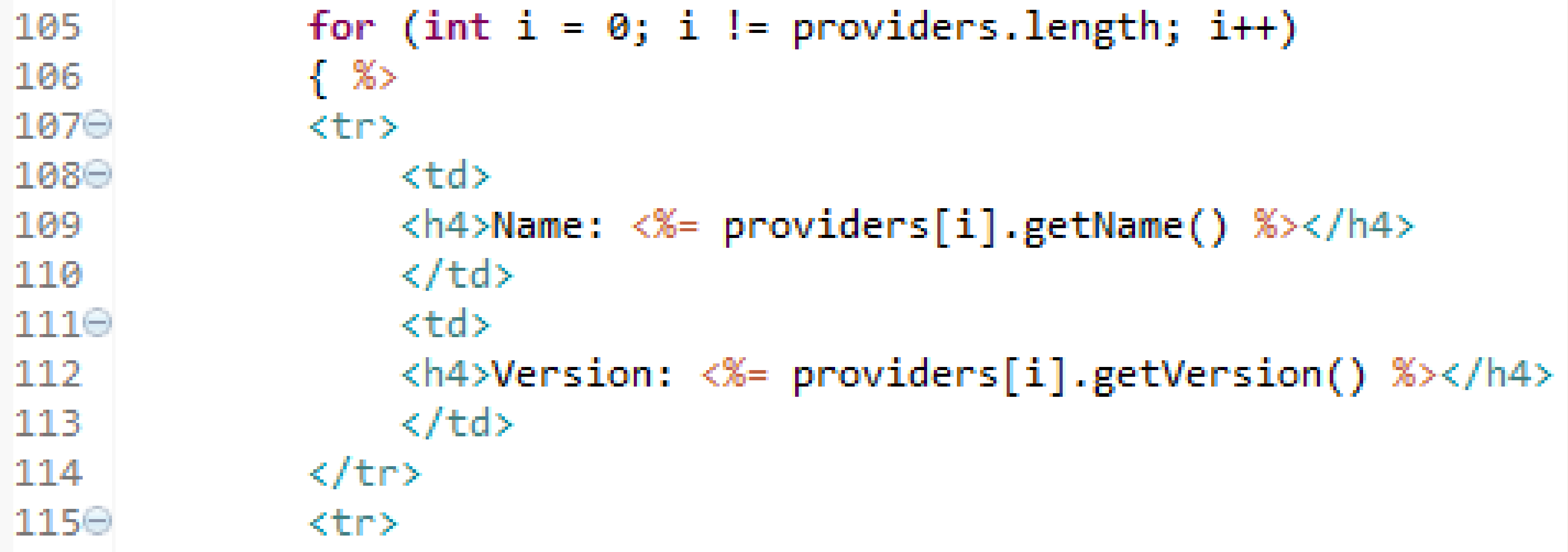
Let's take a closer look.

Google App Engine’s Crypto Providers

A simple JSP-page unveils the secrets:

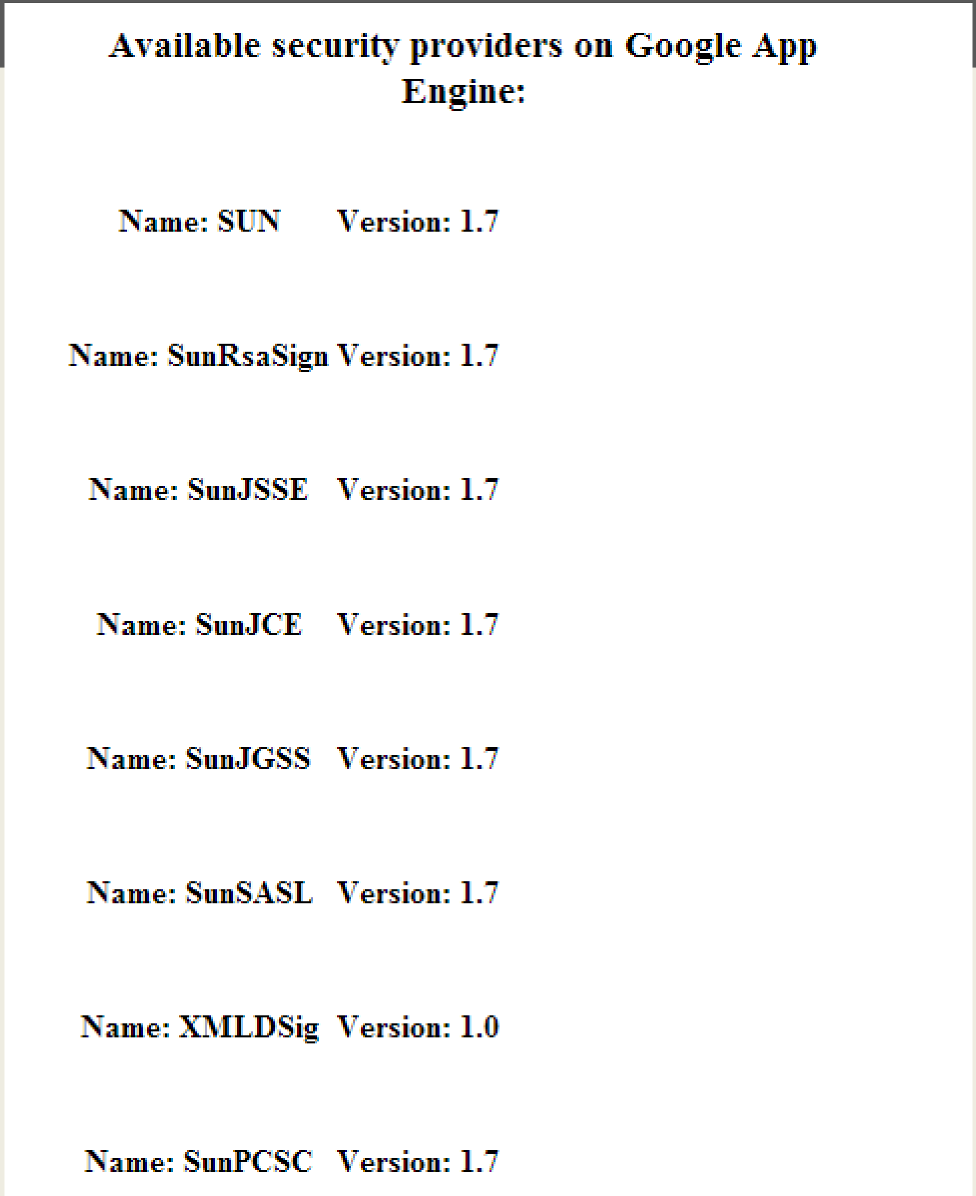


```
    Provider[]  providers = Security.getProviders();
```



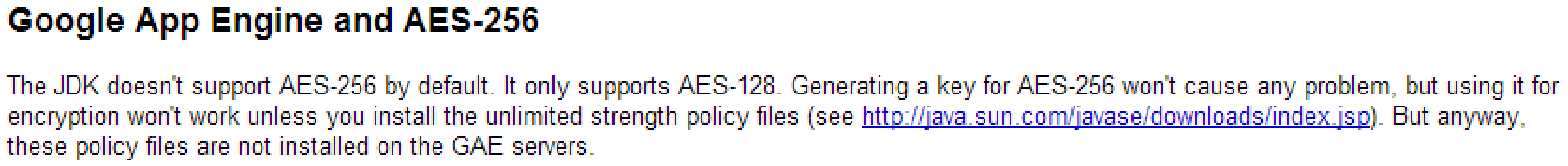
```
        for (int i = 0; i != providers.length; i++)
        { %>
        <tr>
            <td>
            <h4>Name: <%= providers[i].getName() %></h4>
            </td>
            <td>
            <h4>Version: <%= providers[i].getVersion() %></h4>
            </td>
        </tr>
        <tr>
```

Google App Engine's Crypto Providers



```
Available security providers on Google App
                  Engine:

   Name: SUN        Version: 1.7

Name: SunRsaSign Version: 1.7

  Name: SunJSSE   Version: 1.7

   Name: SunJCE    Version: 1.7

  Name: SunJGSS   Version: 1.7

  Name: SunSASL   Version: 1.7

 Name: XMLDSig  Version: 1.0

  Name: SunPCSC   Version: 1.7
```

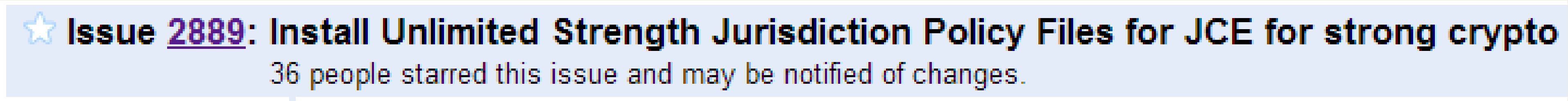
A closer look on encryption



Google App Engine and AES-256

The JDK doesn't support AES-256 by default. It only supports AES-128. Generating a key for AES-256 won't cause any problem, but using it for encryption won't work unless you install the unlimited strength policy files (see http://java.sun.com/javase/downloads/index.jsp). But anyway, these policy files are not installed on the GAE servers.

There is an accepted issue on Google App Engine's Java Runtime



☆ **Issue 2889: Install Unlimited Strength Jurisdiction Policy Files for JCE for strong crypto**

36 people starred this issue and may be notified of changes.

BUT:

Medium Priority

Unchanged since August 2011 ☹

Google App Engine's Crypto Providers

- What do we need?
  - Cryptography Architecture
  - File Access
  - Performance and Scalability
- What do we have already?
  - ✓ Cryptography Architecture
  - ✓ File Access
  - ✓ Performance and Scalability? It's the cloud ;)

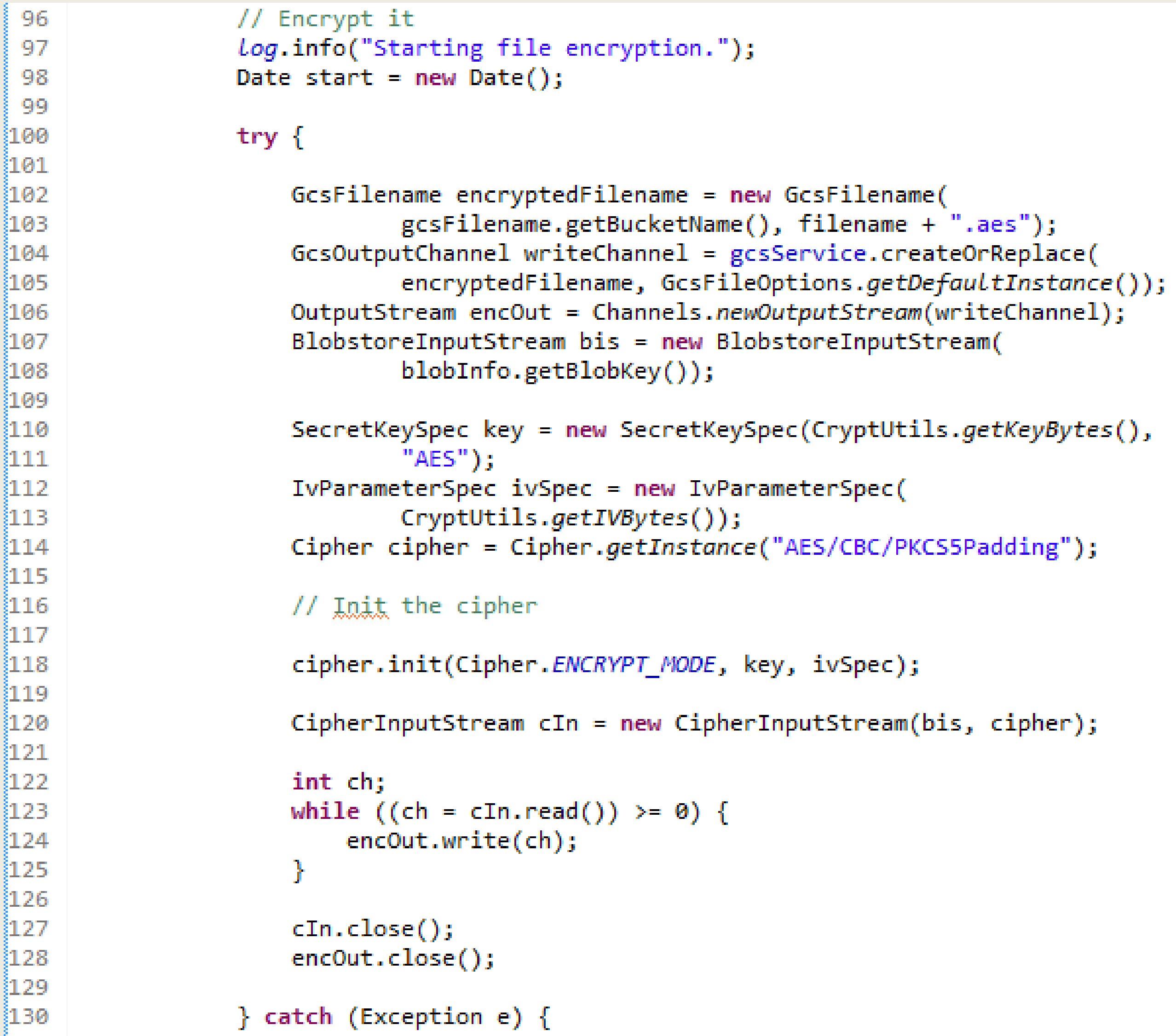
Let’s focus on symmetric encryption

- Symmetric key ciphers are the workhorses of cryptography
  - Used to secure bulk data
  - provide a foundation for message authentication codes
  - provide support for passwordbased encryption
- Security is gained from keeping a shared secret

Symmetric key ciphers in JCA

- JCA (Java Cryptography Architecture)
  - Design principles
    - Implementation independence and interoperability
    - Algorithm independence and extensibility
  - Algorithm independence achieved by crypto engines:
    - `MessageDigest`
    - `Signature`
    - `KeyFactory`
    - `KeyPairGenerator`
    - `Cipher`
  - Different Cryptographic Service Providers

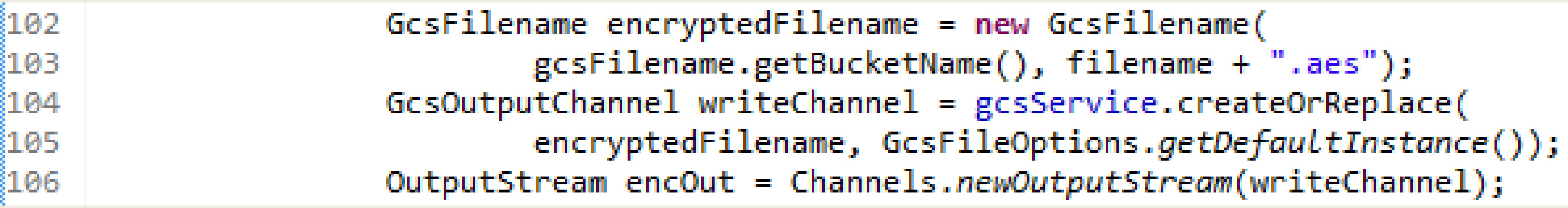
Cloud encryption code



```
            // Encrypt it
            log.info("Starting file encryption.");
            Date start = new Date();

            try {

                GcsFilename encryptedFilename = new GcsFilename(
                        gcsFilename.getBucketName(), filename + ".aes");
                GcsOutputChannel writeChannel = gcsService.createOrReplace(
                        encryptedFilename, GcsFileOptions.getDefaultInstance());
                OutputStream encOut = Channels.newOutputStream(writeChannel);
                BlobstoreInputStream bis = new BlobstoreInputStream(
                        blobInfo.getBlobKey());

                SecretKeySpec key = new SecretKeySpec(CryptUtils.getKeyBytes(),
                        "AES");
                IvParameterSpec ivSpec = new IvParameterSpec(
                        CryptUtils.getIVBytes());
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

                // Init the cipher

                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);

                CipherInputStream cIn = new CipherInputStream(bis, cipher);

                int ch;
                while ((ch = cIn.read()) >= 0) {
                    encOut.write(ch);
                }

                cIn.close();
                encOut.close();

            } catch (Exception e) {
```

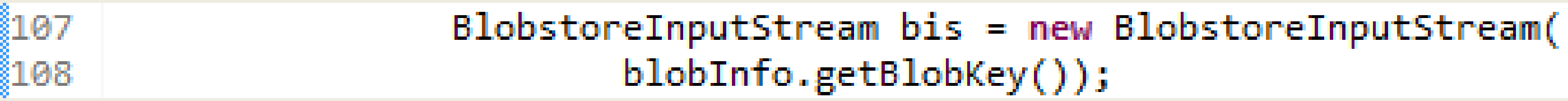
Cloud encryption code

- Setup the encrypted file out stream



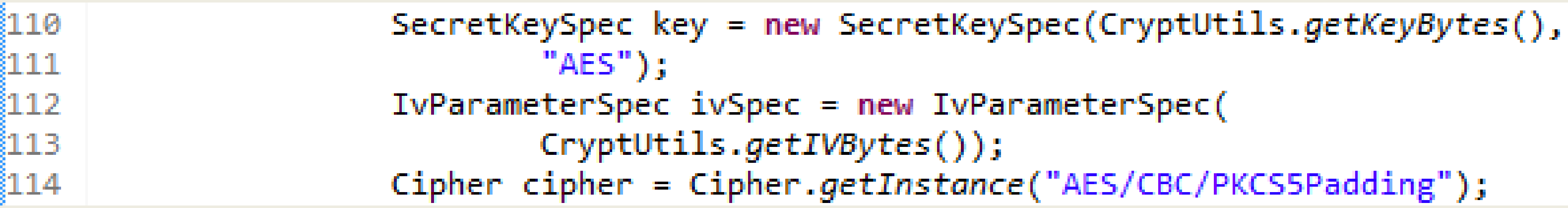
```java
GcsFilename encryptedFilename = new GcsFilename(
        gcsFilename.getBucketName(), filename + ".aes");
GcsOutputChannel writeChannel = gcsService.createOrReplace(
        encryptedFilename, GcsFileOptions.getDefaultInstance());
OutputStream encOut = Channels.newOutputStream(writeChannel);
```

- Setup the clear input stream



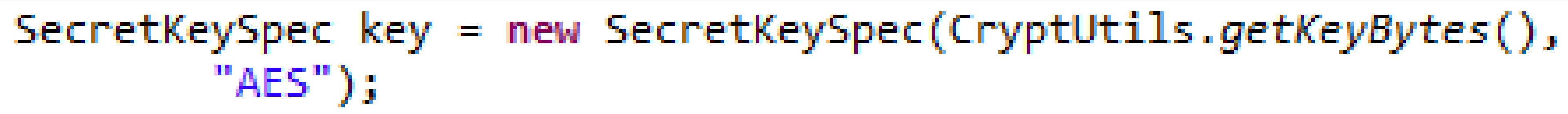
```java
BlobstoreInputStream bis = new BlobstoreInputStream(
        blobInfo.getBlobKey());
```

- Setup the Cipher



```java
SecretKeySpec key = new SecretKeySpec(CryptUtils.getKeyBytes(),
        "AES");
IvParameterSpec ivSpec = new IvParameterSpec(
        CryptUtils.getIVBytes());
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

Cloud encryption code



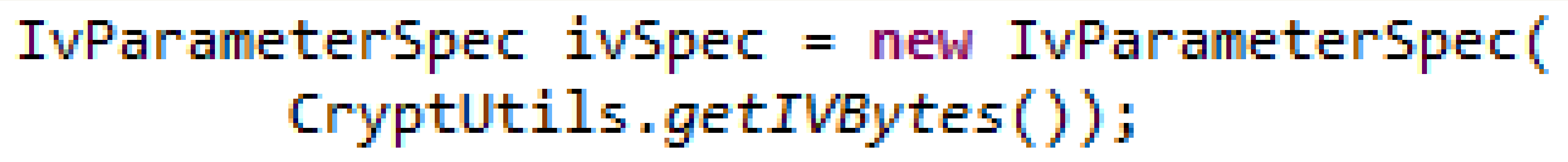
```
SecretKeySpec key = new SecretKeySpec(CryptUtils.getKeyBytes(),
        "AES");
```

SecretKeySpec

provides a simple mechanism to convert bytes into a secret key

IMPORTANT: Does not stop you from using weak keys!

Cloud encryption code



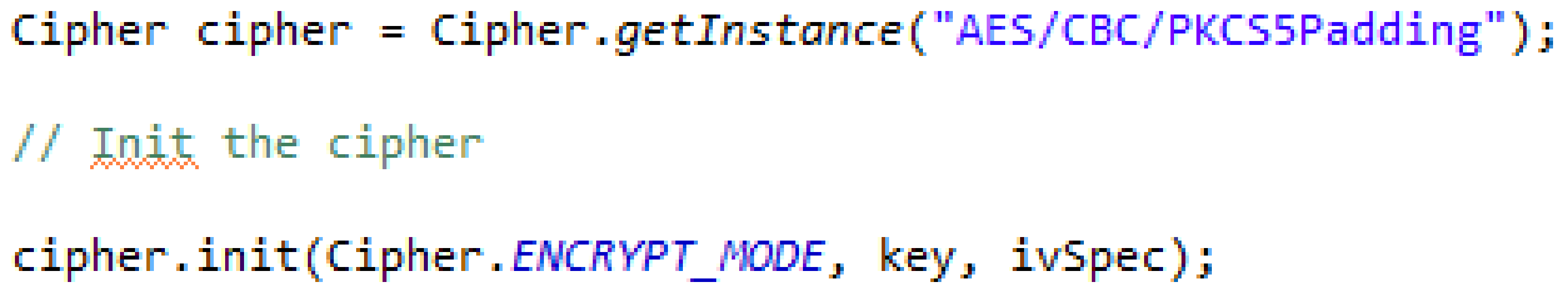
```
IvParameterSpec ivSpec = new IvParameterSpec(
        CryptUtils.getIVBytes());
```

`IvParameterSpec`

is used to perform an XOR operation on the first block of plaintext.

IMPORTANT: This is required for Ciphers in CBC mode. Beyond that, a random IV (Initialization Vector) is crucial for strong cryptography!

Cloud encryption code



```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

// Init the cipher

cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
```

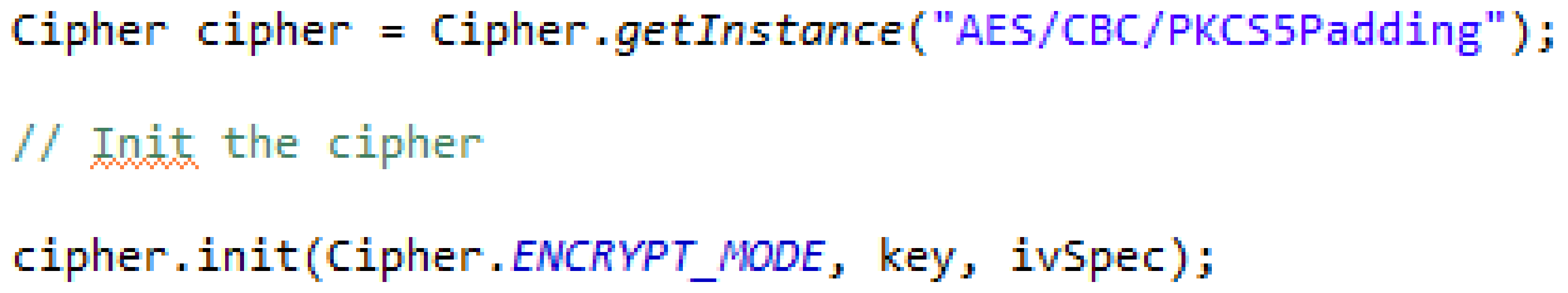
`Cipher`

- `getInstance()`

Consists of three parts:

- – Algorithm (“AES”)
- – Algorithm mode (“CBC”)
- – Padding (“PKCS5Padding”)

Cloud encryption code



```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

// Init the cipher

cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
```

`Cipher`

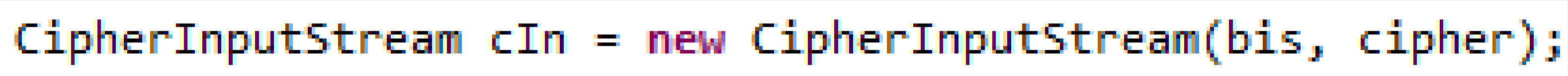
- `init()`

takes (in this case) three arguments:

- Mode (Encrypt or Decrypt)
- Key
- and the IV

IMPORTANT: If you do not init the Cipher, an `IllegalStateException` is thrown.

Cloud encryption code



```
CipherInputStream cIn = new CipherInputStream(bis, cipher);
```

`CipherInputStream`

simply wraps the passed streams and then filters anything read or written to them through the Cipher.

IMPORTANT: Do not forget to close the stream.

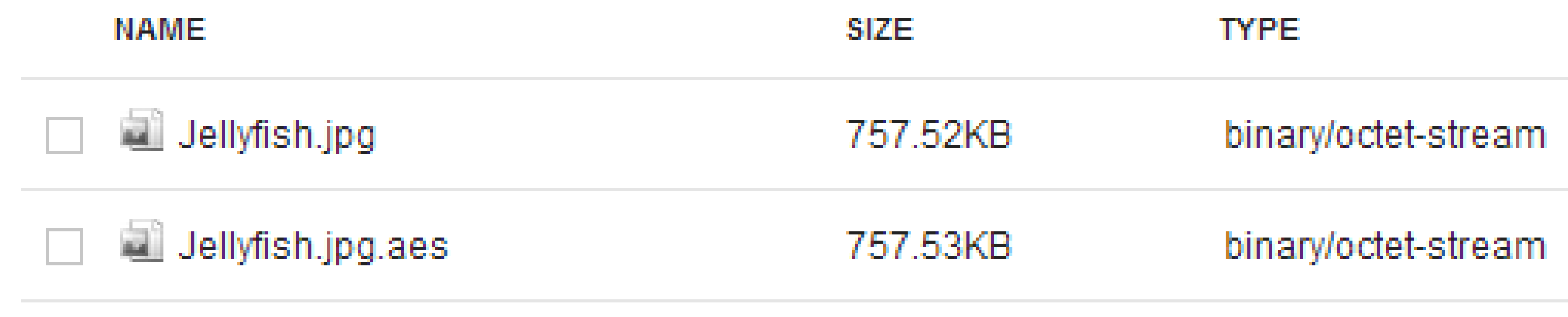
Cloud encryption?

- Not really – this was plain old Java encryption
- So where's the cloud?
  - `BlobstoreInputStream`
  - `GcsInputChannel`
  - `GcsOutputChannel`

`extends java.nio.channels`

Cloud file encrypted

File in the cloud store encrypted ✓

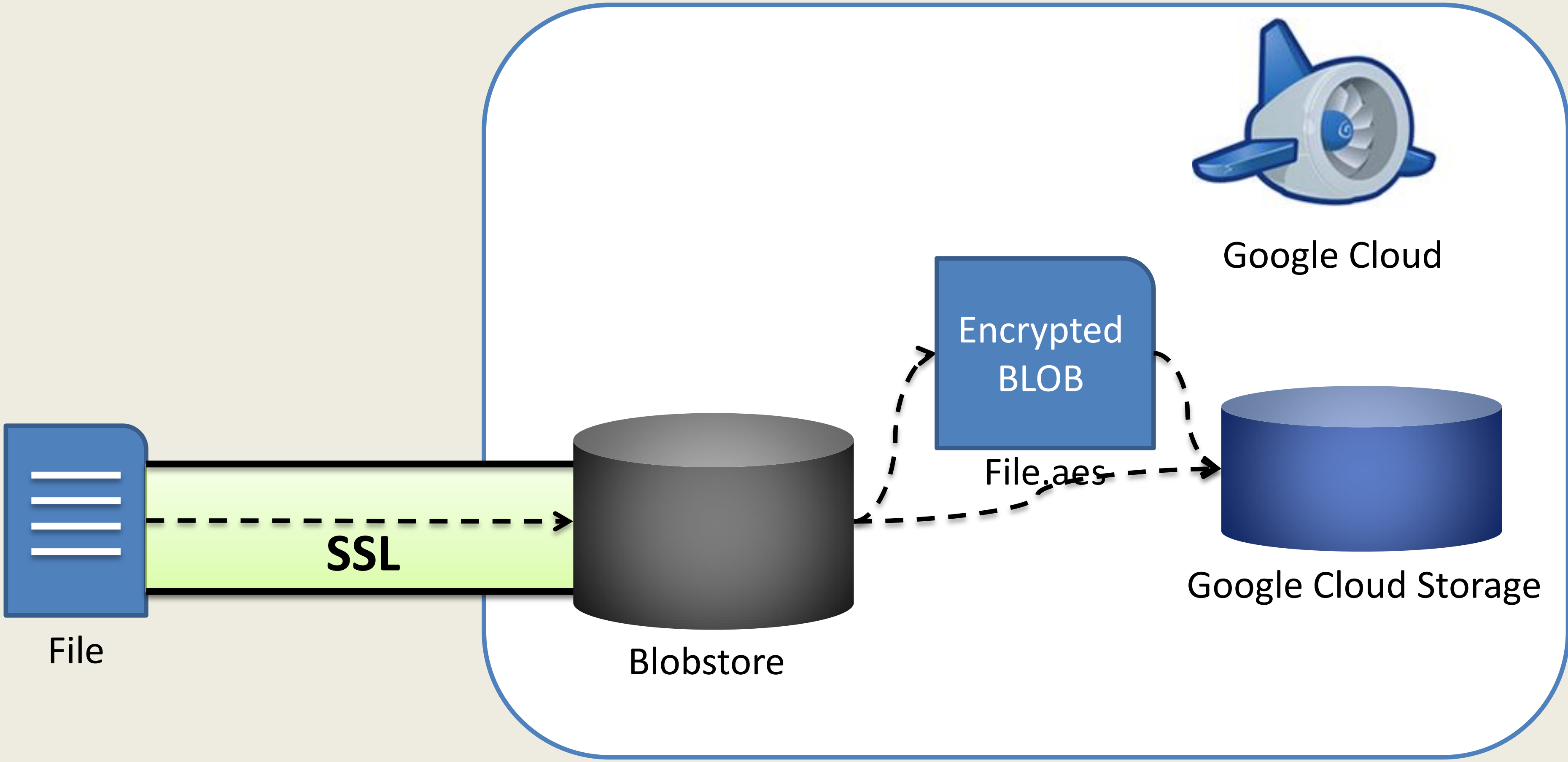


	NAME	SIZE	TYPE
□	Jellyfish.jpg	757.52KB	binary/octet-stream
□	Jellyfish.jpg.aes	757.53KB	binary/octet-stream

Things to do from here:

- Delete plain file
- Manage the keys

Encryption in our example



Where are we now?

1. Drag-and-Drop into the Google Cloud ✓

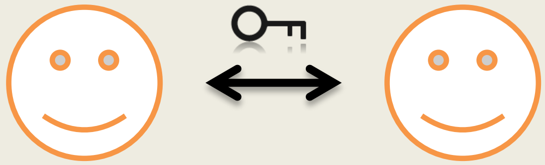
2. SSL and Google App Engine ✓

3. Encrypt your files in the Google Cloud ✓

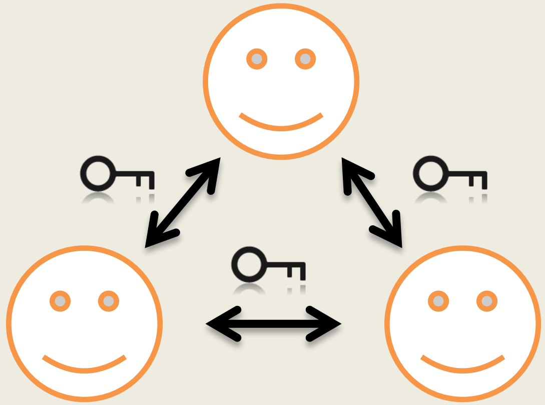
4. Manage the keys – the tricky part

5. Conclusion and outlook

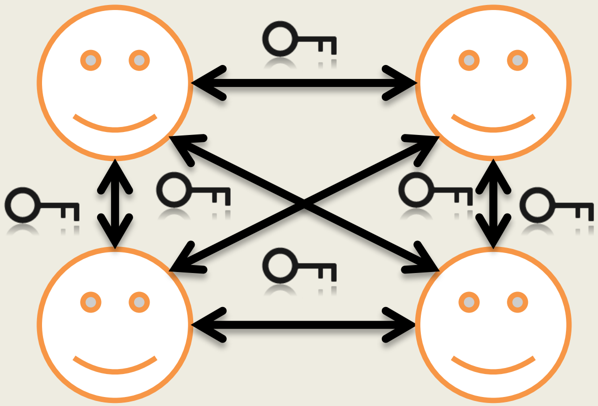
The key exchange problem



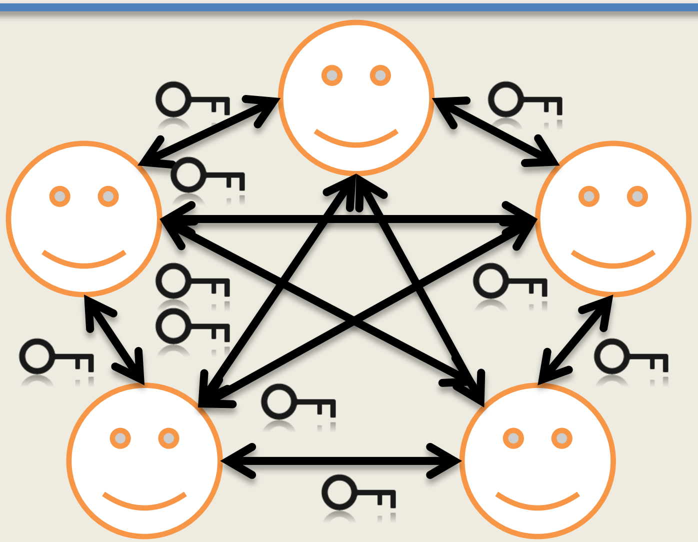
2 Participants, 1 key



3 Participants, 3 keys

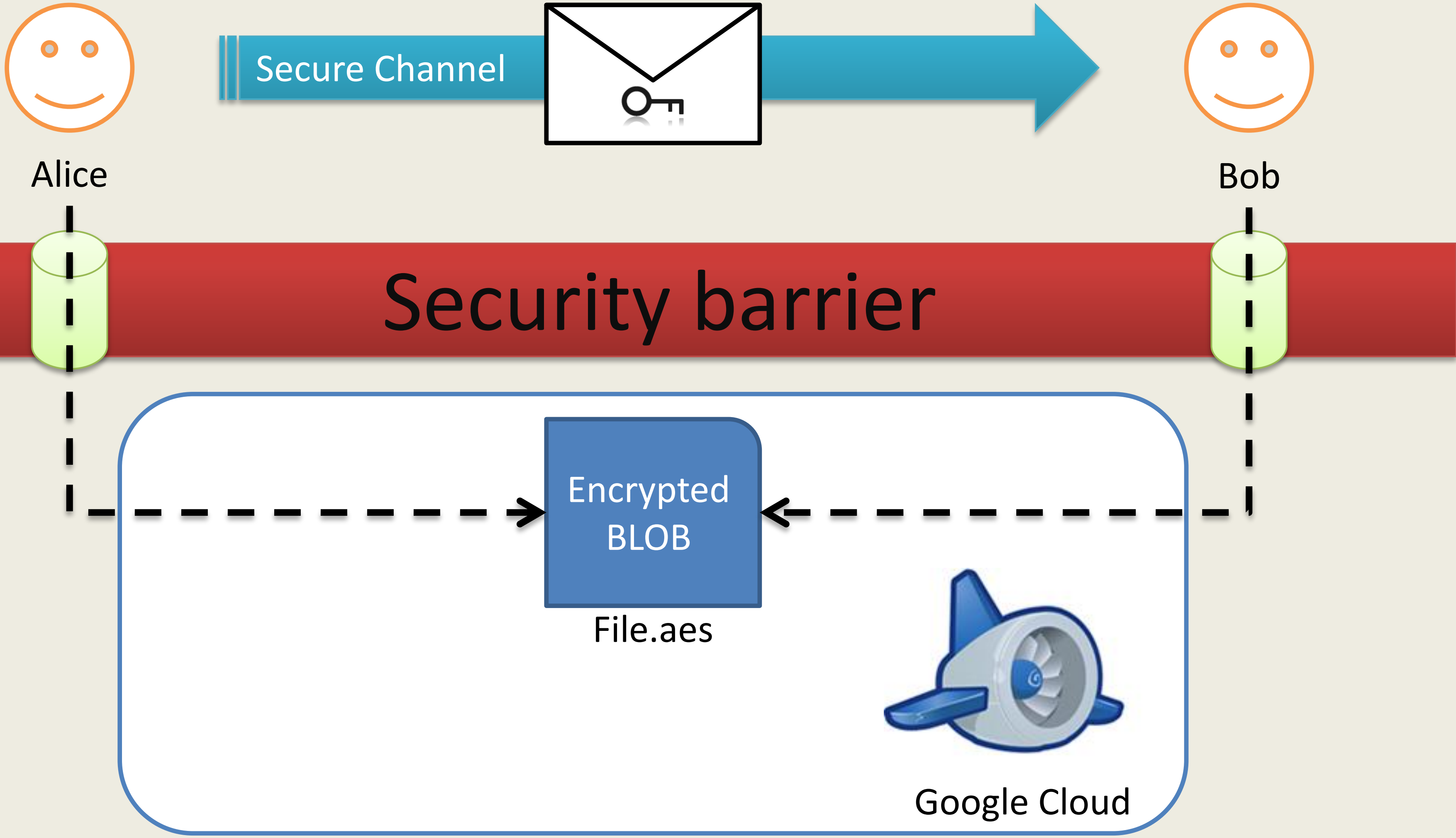


4 Participants, 6 keys

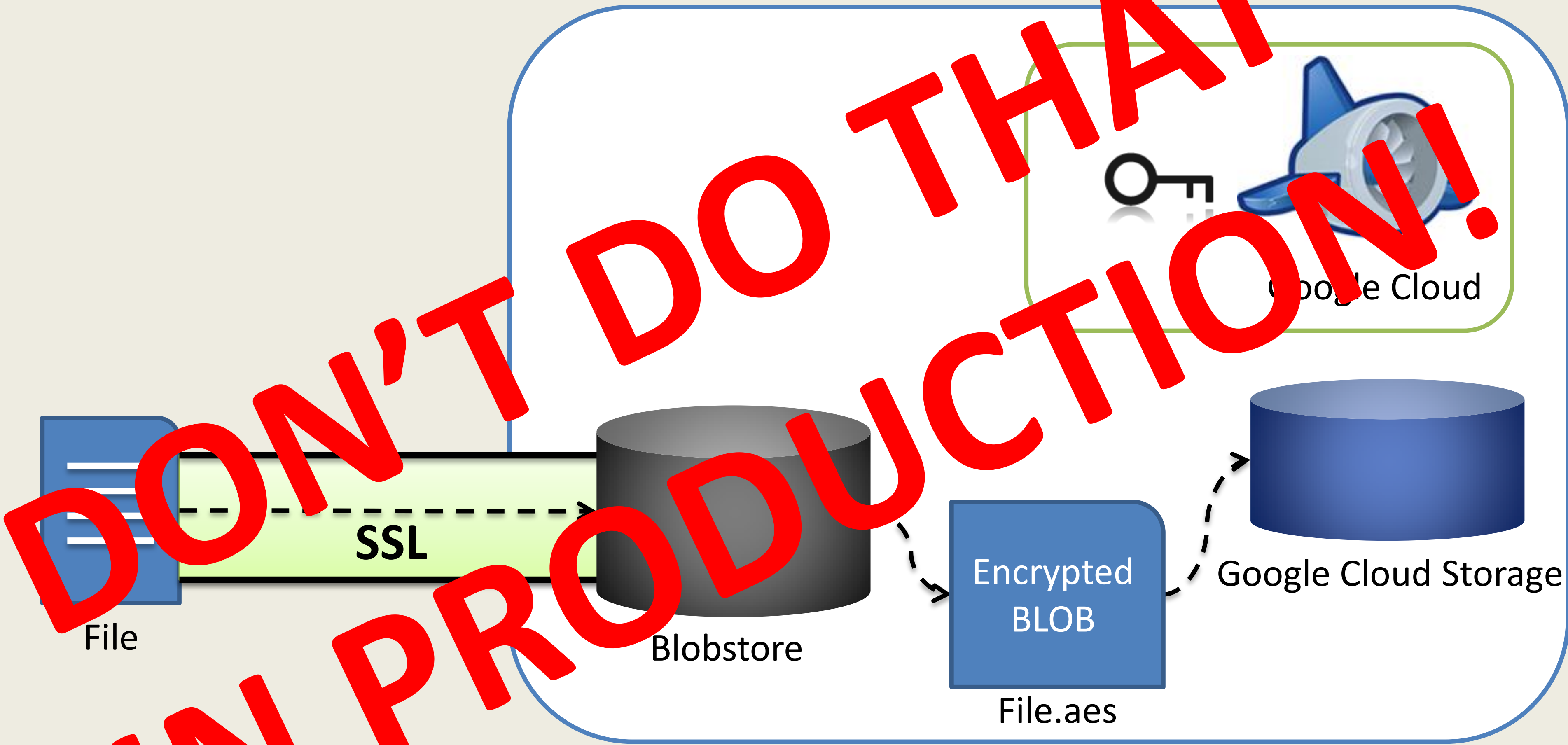


5 Participants, 10 keys

Keep the keys private (and out of the cloud)



Key storage in our example



DON'T DO THAT IN PRODUCTION!

Possible solutions

- Public Key Infrastructure
  - The best approach
  - Performance for encryption is low
  - It’s tricky
- Time and access-controlled Gateways in the Security Barrier
  - A “pragmatic” approach
  - Easy to install
  - But not very secure
- Client side encryption
  - Hard in the browser without plugins
  - Secure if the provider has “zero-knowledge”

Where are we now?

1. Drag-and-Drop into the Google Cloud ✓

2. SSL and Google App Engine ✓

3. Encrypt your files in the Google Cloud ✓

4. Manage the keys – the tricky part ✓

5. Conclusion and outlook

HTML 5

- Lightweight
  - If you are not using plugins
  - Great UX
  - Client-side encryption available, but not standardized
- Great platform
  - W3C File API, FileReader, …
- Many frameworks around, choose wisely!
  - Do not underestimate the research effort

Google App Engine

- The environment is maturing
  - Finally: File-Access is available
  - Architecture is gaining complexity
- Pragmatic alternative to other Cloud stacks
  - “Hack and Deploy” – approach

Encryption in the Cloud

- still in it’s infancy
- expect many new approaches coming up
- Everybody of us uses the cloud
  - Providers tend to add encryption as “buzz” word
  - Take a close look at what it REALLY means
  - Transparency needs to be improved DRAMATICALLY!
- Key management is crucial

expressFlow (I’ll keep it short)

1. Drop your files

2. Encrypt your files

3. Store files in your cloud

Drag & Drop Files
from your computer
into the browser ...
Automagically
encrypt files with
expressFlow ...
Dropbox
Google Drive
Box
Put encrypted files
in your favorite
cloud store ...

- Upcoming later this year
  - Paranoid version (subscribe to get a preview)
  - Mobile version
  - Enterprise-level “in-cloud” licenses for:
    - Google App Engine
    - Amazon Web services

Thank you!

Questions?

Now!

or via email: martin@expressflow.com

Fork us on GitHub

Likes us:

Follow us:

Add us: