Secrets at Planet-Scale: Engineering the Internal Google Key Management System (KMS)

Anvita Pandit

Google LLC

QCon San Francisco 2019, Nov 11-13
Anvita Pandit

- Software engineer in Data Protection / Security and Privacy org in Google for 2 years.
- Engineering Resident.
- DEFCON 2019 Biohacking village: co-presented “Hacking Race” workshop with @HerroAnneKim
Not the Google Cloud KMS
Agenda

1. Why use a KMS?
2. Essential product features
3. Walkthrough of encrypted storage use case
4. System specs and architectural decisions
5. Walkthrough of an outage
6. More architecture!
7. Challenge: safe key rotation
The Great Gmail Outage of 2014

https://googleblog.blogspot.com/2014/01/todays-outage-for-several-google.html
Why Use a KMS?
Why Use a KMS?

Core motivation: code needs secrets!
Why Use a KMS?

Core motivation: code needs secrets!

Secrets like:

- Database passwords, third party API and OAuth tokens
- Cryptographic keys used for data encryption, signing, etc
Why Use a KMS?

Core motivation: code needs secrets!

Where?
Why Use a KMS?

Core motivation: code needs secrets!

Where?
● In code repository?
Showing 335,659 available commit results

- **swczphyo/python_castle**
  - update and remove passwords
  - swczphyo committed 2 days ago

- **hsunches/app-iot-service**
  - remove trim for password
  - Stan committed 6 days ago

- **clmay/dotfiles**
  - Remove master-password cask
  - clmay committed 13 days ago

- **Rafase282/mangadb-front**
  - Adds support for password reset (#49)
  - Rafase282 committed 9 days ago

- **bmazeju/appsec**
  - removing second password confirmation requirement
  - bmazeju committed 12 days ago

https://github.com/search?utf8=%E2%9C%93&q=remove+password&type=Commits&ref=searchresults
Why Use a KMS?

Core motivation: code needs secrets!

Where?
- In code repository?
- On production hard drives?
Why Use a KMS?

Core motivation: code needs secrets!

Where?
- In code repository?
- On production hard drives?

Alternative:
- Use a KMS!
Centralized Key Management

Solves *key* problems for everybody.
Centralized Key Management

Solves key problems for everybody.

Offers:

- Separate management of key-handling code
Centralized Key Management
Solves key problems for everybody.

Offers:

- Separate management of key-handling code
- Separation of trust
Centralized Key Management

Solves *key* problems for everybody
Centralized Key Management

Solves *key* problems for everybody

1. Access control lists (ACLs)
Centralized Key Management
Solves key problems for everybody

1. Access control lists (ACLs)
   ● Who is allowed to use the key? Who is allowed to make updates to the key configuration?
Centralized Key Management

Solves key problems for everybody

1. Access control lists (ACLs)
   ● Who is allowed to use the key? Who is allowed to make updates to the key configuration?
   ● Identities are specified with the internal authentication system (see ALTS)
Centralized Key Management
Solves key problems for everybody.

2. Auditing aka Who touched my keys?
Centralized Key Management
Solves key problems for everybody.

2. Auditing aka Who touched my keys?
   - Binary verification
Centralized Key Management
Solves key problems for everybody.

2. Auditing aka Who touched my keys?
   - Binary verification
   - Logging (but not the secrets!)
Data is uploaded to Google

Data is chunked and each chunk is encrypted with its own key

Chunks are distributed across Google's storage infrastructure
Data is uploaded to Google

Data is chunked and each chunk is encrypted with its own key

Chunks are distributed across Google’s storage infrastructure
Data is uploaded to Google

Data is chunked and each chunk is encrypted with its own key

Chunks are distributed across Google's storage infrastructure
Data is uploaded to Google

Data is chunked and each chunk is encrypted with its own key

Chunks are distributed across Google’s storage infrastructure

Data chunk encrypted with DEK and stored with wrapped DEK

Request for unwrapping DEK

Return unwrapped DEK
Google’s Root of Trust

**Storage Systems (Millions)**
Data encrypted with data keys (DEKs)

- **KMS (Tens of Thousands)**
  Master keys and passwords are stored in KMS

- **Root KMS (Hundreds)**
  KMS is protected with a KMS master key in Root KMS

- **Root KMS master key distributor (Hundreds)**
  Root KMS master key is distributed in memory

- **Physical safes (a few)**
  Root KMS master key is backed up on hardware devices
Google’s Root of Trust

**Storage Systems (Millions)**
Data encrypted with data keys (DEKs)

**KMS (Tens of Thousands)**
Master keys and passwords are stored in KMS

**Root KMS (Hundreds)**
KMS is protected with a KMS master key in Root KMS

**Root KMS master key distributor (Hundreds)**
Root KMS master key is distributed in memory

**Physical safes (a few)**
Root KMS master key is backed up on hardware devices
Google’s Root of Trust

Storage Systems (Millions)
Data encrypted with data keys (DEKs)

KMS (Tens of Thousands)
Master keys and passwords are stored in KMS

Root KMS (Hundreds)
KMS is protected with a KMS master key in Root KMS

Root KMS master key distributor (Hundreds)
Root KMS master key is distributed in memory

Physical safes (a few)
Root KMS master key is backed up on hardware devices
Google’s Root of Trust

Storage Systems (Millions)
Data encrypted with data keys (DEKs)

KMS (Tens of Thousands)
Master keys and passwords are stored in KMS

Root KMS (Hundreds)
KMS is protected with a KMS master key in Root KMS

Root KMS master key distributor (Hundreds)
Root KMS master key is distributed in memory

Physical safes (a few)
Root KMS master key is backed up on hardware devices
Google’s Root of Trust

**Storage Systems (Millions)**
Data encrypted with data keys (DEKs)

**KMS (Tens of Thousands)**
Master keys and passwords are stored in KMS

**Root KMS (Hundreds)**
KMS is protected with a KMS master key in Root KMS

**Root KMS master key distributor (Hundreds)**
Root KMS master key is distributed in memory

**Physical safes (a few)**
Root KMS master key is backed up on hardware devices
## Design Requirements

<table>
<thead>
<tr>
<th>Category</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>5 nines =&gt; 99.999% of requests are served</td>
</tr>
<tr>
<td>Latency</td>
<td>99% of requests are served &lt; 10 ms</td>
</tr>
<tr>
<td>Scalability</td>
<td>Planet-scale!</td>
</tr>
<tr>
<td>Security</td>
<td>Effortless key rotation</td>
</tr>
</tbody>
</table>
Decisions, decisions

- Not an encryption/decryption service.
Decisions, decisions

- Not an encryption/decryption service.
- Not a traditional database
Decisions, decisions

- Not an encryption/decryption service.
- Not a traditional database
- Key wrapping
- Stateless serving
Key Wrapping
Key Wrapping

- Fewer centrally-managed keys improves availability but requires more trust in the client
Stateless Serving

Insight: At the KMS layer, key material is not mutable state.

Immutable key material + key wrapping

  ==> Stateless server ==> Trivial **scaling**

Keys in RAM ==> **Low latency** serving
What Could Go Wrong?
The Great Gmail Outage of 2014

We're sorry, but your Gmail account is temporarily unavailable. We apologize for the inconvenience and suggest trying again in a few minutes. You can view the Apps Status Dashboard for the current status of the service.

If the issue persists, please visit the Gmail Help Center »

Try Again Sign Out

Show Detailed Technical Info

https://googleblog.blogspot.com/2014/01/todays-outage-for-several-google.html
Each team maintains their own KMS configurations, all stored in Google’s monolithic repo:

Source Repository (holds encrypted configs)

Individual Team Config Changes

Which get automatically merged into a combined config file

Which is distributed daily to all KMS servers for serving.

Normal Operation

Normal Operation

Merging Problem

Truncated Config

Update Data Pusher

KMS Server

KMS

Local Config

Many KMS Servers

All Local Configs

A bad config pushed globally means a global outage

Client

Client

Sees incorrect image of source repo 😡
Lessons Learned

The KMS had become

- a single point of failure
- a startup dependency for services
- often a runtime dependency

==> KMS Must Not Fail Globally
KMS Must Not Fail Globally

- No more all-at-once global rollout of binaries and configuration
- Regional failure isolation and client isolation
- Minimize dependencies
Google KMS Current Stats:

- No downtime since the Gmail outage in 2014 January: $99.9999\%$
- $99.9\%$ of requests are served $< 6 \text{ ms}$
- $\sim10^7$ requests/sec ($\sim10$ M QPS)
- $\sim10^4$ processes & cores
Challenge: Safe Key Rotation
Make It Easy To Rotate Keys

- Key compromise
  - Also requires access to cipher text
Make It Easy To Rotate Keys

- Key compromise
  - Also requires access to cipher text
- Broken ciphers
  - Access to cipher text is enough
Make It Easy To Rotate Keys

- Key compromise
  - Also requires access to cipher text

- Broken ciphers
  - Access to cipher text is enough

- Rotating keys limits the window of vulnerability
Make It Easy To Rotate Keys

- Key compromise
  - Also requires access to cipher text
- Broken ciphers
  - Access to cipher text is enough
- Rotating keys limits the window of vulnerability
- *But* rotating keys means there is potential for data loss
Robust Key Rotation at Scale - 0

Goals

1. KMS users design with rotation in mind
2. Using multiple key versions is no harder than using a single key
3. Very hard to lose data
Robust Key Rotation at Scale - 1

Goal #1: KMS users design with rotation in mind

● Users choose
  ○ Frequency of rotation: e.g. every 30 days
  ○ TTL of cipher text: e.g. 30,90,180 days, 2 years, etc.
Robust Key Rotation at Scale - 1

Goal #1: KMS users design with rotation in mind

- Users choose
  - Frequency of rotation: e.g. every 30 days
  - TTL of cipher text: e.g. 30, 90, 180 days, 2 years, etc.

- KMS guarantees ‘Safety Condition’
  - All ciphertext produced within the TTL can be deciphered using a keyset in the KMS.
Robust Key Rotation at Scale - 2

Goal #2: Using multiple key versions is no harder than using a single key
Robust Key Rotation at Scale - 2

Goal #2: Using multiple key versions is no harder than using a single key

- Tightly integrated with Google's standard cryptographic libraries: see Tink
Robust Key Rotation at Scale - 2

Goal #2: Using multiple key versions is no harder than using a single key

- Tightly integrated with Google's standard cryptographic libraries: see Tink
  - Keys support **multiple key versions**
  - Each of which can be a different cipher
Robust Key Rotation at Scale - 3

Goal #3: Very hard to lose data

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>SFR</td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

A - Active
P - Primary
SFR - Scheduled for Revocation
Robust Key Rotation at Scale - 3

Goal #3: Very hard to lose data

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>SFR</td>
</tr>
<tr>
<td>V2</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

A - Active
P - Primary
SFR - Scheduled for Revocation
Robust Key Rotation at Scale - 3

Goal #3: Very hard to lose data

<table>
<thead>
<tr>
<th>V1</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>SFR</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V2</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>
Robust Key Rotation at Scale - 3

Goal #3: Very hard to lose data

<table>
<thead>
<tr>
<th>Time</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>SFR</td>
</tr>
<tr>
<td>V2</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

A - Active
P - Primary
SFR - Scheduled for Revocation
Recap: Key Rotation
Recap: Key Rotation

- Presents an availability vs security tradeoff
Recap: Key Rotation

- Presents an availability vs security tradeoff
- KMS
  - Derives the number of key versions to retain
Recap: Key Rotation

- Presents an availability vs security tradeoff
- KMS
  - Derives the number of key versions to retain
  - Adds/Promotes/Demotes/Deletes Key Versions over time
Google KMS - Summary

Implementing encryption at scale required highly available key management.

At Google’s scale this means 5 9s of availability.

To achieve all requirements, we use several strategies:

- Best practices for change management and staged rollouts
- Minimize dependencies and aggressively defend against their unavailability
- Isolate by region & client type
- Combine immutable keys + wrapping to achieve scale
- A declarative API for key rotation
We Are Hiring!

anvita@google.com
Further Reading

■ Google Cloud Encryption at Rest whitepaper: https://cloud.google.com/security/encryption-at-rest/default-encryption/


+ Infographic https://cloud.withgoogle.com/infrastructure/data-encryption/step-7

■ Tink cryptographic library https://github.com/google/tink

Bonus Content
Challenge: Data Integrity
Causes of Bit Errors

- Corruption in transit as NICs (network cards) twiddle bits.
- Corruption in memory from broken CPUs
- Cosmic rays flip bits in DRAM
- [not an exhaustive list]
Hardware Faults

- Crypto provides leverage
- Key material corruption can render large chunks of data unusable.
Software Mitigations

○ Verify correctness of crypto operations at start of a process
  ■ During a request, after using the KEK to wrap a DEK and before responding to the customer, we unwrap the same DEK
  ■ Storage services
    ● Read back plain text after writing encrypted data blocks
    ● Replicate/parity protect at a higher layer
Key Sensitivity Annotations
Key Sensitivity Annotations

Users determine the consequence if their keys were to be compromised using the CIA triad
Sensitivity Annotations

Users determine the consequence if their keys were to be compromised using the CIA triad:

- Confidentiality
- Integrity
- Availability
Sensitivity Annotations

- Each consequence has corresponding policy recommendations
- For example, only a verifiably built program can contact a key that could leak user data.