Google Cloud Security Whitepapers

March 2018

Google Cloud Infrastructure Security Design Overview

Encryption at Rest in Google Cloud

Encryption in Transit in Google Cloud

Application Layer Transport Security in Google Cloud

Google Cloud

Google Cloud

Table of Contents

Google Infrastructure Security Design Overview

A technical whitepaper from Google Cloud

Google Cloud

Google Cloud

Table of Contents

Google Cloud

The content contained herein is correct as of January 2017, and represents the status quo as of the time it was written. Google's security policies and systems may change going forward, as we continually improve protection for our customers.

CIO-level summary

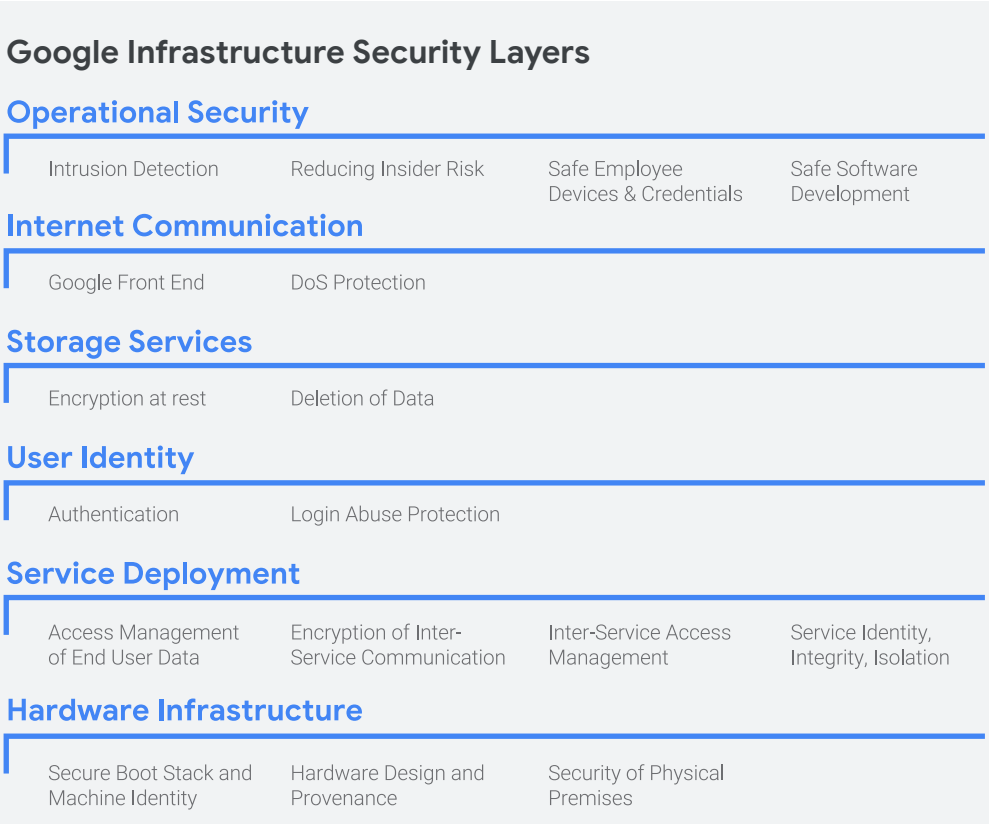
- Google has a global scale technical infrastructure designed to provide security through the entire information processing lifecycle at Google. This infrastructure provides secure deployment of services, secure storage of data with end user privacy safeguards, secure communications between services, secure and private communication with customers over the internet, and safe operation by administrators.
- Google uses this infrastructure to build its internet services, including both consumer services such as Search, Gmail, and Photos, and enterprise services such as G Suite and Google Cloud Platform.
- The security of the infrastructure is designed in progressive layers starting from the physical security of data centers, continuing on to the security of the hardware and software that underlie the infrastructure, and finally, the technical constraints and processes in place to support operational security.
- Google invests heavily in securing its infrastructure with many hundreds of engineers dedicated to security and privacy distributed across all of Google, including many who are recognized industry authorities.

Introduction

This document gives an overview of how security is designed into Google's technical infrastructure. This global scale infrastructure is designed to provide security through the entire information processing lifecycle at Google. This infrastructure provides secure deployment of services, secure storage of data with end user privacy safeguards, secure communications between services, secure and private communication with customers over the internet, and safe operation by administrators.

Google uses this infrastructure to build its internet services, including both consumer services such as Search, Gmail, and Photos, and enterprise services such as G Suite and Google Cloud Platform.

We will describe the security of this infrastructure in progressive layers starting from the physical security of our data centers, continuing on to how the hardware and software that underlie the infrastructure are secured, and finally, describing the technical constraints and processes in place to support operational security.



[Figure 1]
Google Infrastructure Security Layers

The various layers of security starting from hardware infrastructure at the bottom layer up to operational security at the top layer. The contents of each layer are described in detail in the paper.

Secure Low Level Infrastructure

In this section we describe how we secure the lowest layers of our infrastructure, ranging from the physical premises to the purpose-built hardware in our data centers to the low-level software stack running on every machine.

Security of Physical Premises

Google designs and builds its own data centers, which incorporate multiple layers of physical security protections. Access to these data centers is limited to only a very small fraction of Google employees. We use multiple physical security layers to protect our data center floors and use technologies like biometric identification, metal detection, cameras, vehicle barriers, and laser-based intrusion detection systems. Google additionally hosts some servers in third-party data centers, where we ensure that there are Google-controlled physical security measures on top of the security layers provided by the data center operator. For example, in such sites we may operate independent biometric identification systems, cameras, and metal detectors.

Hardware Design and Provenance

A Google data center consists of thousands of server machines connected to a local network. Both the server boards and the networking equipment are custom-designed by Google. We vet component vendors we work with and choose components with care, while working with vendors to audit and validate the security properties provided by the components. We also design custom chips, including a hardware security chip that is currently being deployed on both servers and peripherals. These chips allow us to securely identify and authenticate legitimate Google devices at the hardware level.

Secure Boot Stack and Machine Identity

Google server machines use a variety of technologies to ensure that they are booting the correct software stack. We use cryptographic

A Google data center consists of thousands of server machines connected to a local network. Both the server boards and the networking equipment are custom designed by Google.

signatures over low-level components like the BIOS, bootloader, kernel, and base operating system image. These signatures can be validated during each boot or update. The components are all Google-controlled, built, and hardened. With each new generation of hardware we strive to continually improve security: for example, depending on the generation of server design, we root the trust of the boot chain in either a lockable firmware chip, a microcontroller running Google-written security code, or the above mentioned Google-designed security chip.

Each server machine in the data center has its own specific identity that can be tied to the hardware root of trust and the software with which the machine booted. This identity is used to authenticate API calls to and from low-level management services on the machine.

Google has authored automated systems to ensure servers run up-to-date versions of their software stacks (including security patches), to detect and diagnose hardware and software problems, and to remove machines from service if necessary.

Secure Service Deployment

We will now go on to describe how we go from the base hardware and software to ensuring that a service is deployed securely on our infra-structure. By 'service' we mean an application binary that a developer wrote and wants to run on our infrastructure, for example, a Gmail SMTP server, a BigTable storage server, a YouTube video transcoder, or an App Engine sandbox running a customer application. There may be thousands of machines running copies of the same service to handle the required scale of the workload. Services running on the infrastructure are controlled by a cluster orchestration service called Borg.

As we will see in this section, the infrastructure does not assume any trust between services running on the infrastructure. In other words, the infrastructure is fundamentally designed to be multi-tenant.

Service Identity, Integrity, and Isolation

We use cryptographic authentication and authorization at the application layer for inter-service communication. This provides strong access control at an abstraction level and granularity that administrators and services can naturally understand.

We do not rely on internal network segmentation or firewalling as our primary security mechanisms, though we do use ingress and egress filtering at various points in our network to prevent IP spoofing as a further security layer. This approach also helps us to maximize our network's performance and availability.

Each service that runs on the infrastructure has an associated service account identity. A service is provided cryptographic credentials that it can use to prove its identity when making or receiving remote procedure calls (RPCs) to other services. These identities are used by clients to ensure that they are talking to the correct intended server, and by servers to limit access to methods and data to particular clients.

Google's source code is stored in a central repository where both current and past versions of the service are auditable. The infrastructure can additionally be configured to require that a service's binaries be built from specific reviewed, checked in, and tested source code. Such code reviews require inspection and approval from at least one engineer other than the author, and the system enforces that code modifications to any system must be approved by the owners of that system. These requirements limit the ability of an insider or adversary to make malicious modifications to source code and also provide a forensic trail from a service back to its source.

We have a variety of isolation and sandboxing techniques for protecting a service from other services running on the same machine. These techniques include normal Linux user separation, language and kernel-based sandboxes, and hardware virtualization. In general, we use more layers of isolation for riskier workloads; for example, when running complex file format converters on user-supplied data or when running user supplied code for products like Google App Engine or Google Compute Engine. As an extra security boundary,

We use cryptographic authentication and authorization at the application layer for inter-service communication. This provides strong access control at an abstraction level and granularity that administrators and services can naturally understand.

we enable very sensitive services, such as the cluster orchestration service and some key management services, to run exclusively on dedicated machines.

Inter-Service Access Management

The owner of a service can use access management features provided by the infrastructure to specify exactly which other services can communicate with it. For example, a service may want to offer some APIs solely to a specific whitelist of other services. That service can be configured with the whitelist of the allowed service account identities and this access restriction is then automatically enforced by the infrastructure.

The owner of a service can use access management features provided by the infrastructure to specify exactly which other services can communicate with it.

Google engineers accessing services are also issued individual identities, so services can be similarly configured to allow or deny their accesses. All of these types of identities (machine, service, and employee) are in a global name space that the infrastructure maintains. As will be explained later in this document, end user identities are handled separately.

The infrastructure provides a rich identity management workflow system for these internal identities including approval chains, logging, and notification. For example, these identities can be assigned to access control groups via a system that allows two party-control where one engineer can propose a change to a group that another engineer (who is also an administrator of the group) must approve. This system allows secure access management processes to scale to the thousands of services running on the infrastructure.

In addition to the automatic API-level access control mechanism, the infrastructure also provides services the ability to read from central ACL and group databases so that they can implement their own custom, fine-grained access control where necessary.

Encryption of Inter-Service Communication

Beyond the RPC authentication and authorization capabilities discussed in the previous sections, the infrastructure also provides cryptographic privacy and integrity for RPC data on the network.

To provide these security benefits to other application layer protocols such as HTTP, we encapsulate them inside our infrastructure RPC mechanisms. In essence, this gives application layer isolation and removes any dependency on the security of the network path. Encrypted inter-service communication can remain secure even if the network is tapped or a network device is compromised.

Services can configure the level of cryptographic protection they want for each infrastructure RPC (e.g. only configure integrity-level protection for low value data inside data centers). To protect against sophisticated adversaries who may be trying to tap our private WAN links, the infrastructure automatically encrypts all infrastructure RPC traffic which goes over the WAN between data centers, without requiring any explicit configuration from the service. We have started to deploy hardware cryptographic accelerators that will allow us to extend this default encryption to all infrastructure RPC traffic inside our data centers.

To protect against sophisticated adversaries who may be trying to tap our private WAN links, the infrastructure automatically encrypts all infrastructure RPC traffic which goes over the WAN between data centers.

Access Management of End User Data

A typical Google service is written to do something for an end user. For example, an end user may store their email on Gmail. The end user's interaction with an application like Gmail spans other services within the infrastructure. So for example, the Gmail service may call an API provided by the Contacts service to access the end user's address book.

We have seen in the preceding section that the Contacts service can be configured such that the only RPC requests that are allowed are from the Gmail service (or from any other particular services that the Contacts service wants to allow).

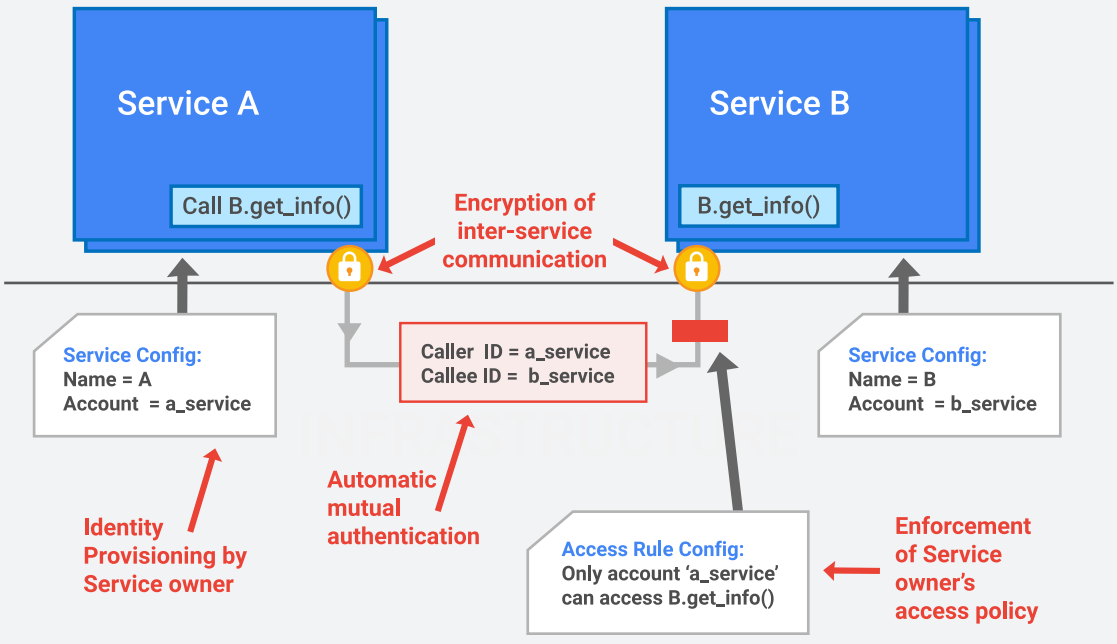
This, however, is still a very broad set of permissions. Within the scope of this permission the Gmail service would be able to request the contacts of any user at any time.

Since the Gmail service makes an RPC request to the Contacts service on behalf of a particular end user, the infrastructure provides a capability for the Gmail service to present an "end user permission ticket" as part of the RPC. This ticket proves that the Gmail service is

currently servicing a request on behalf of that particular end user. This enables the Contacts service to implement a safeguard where it only returns data for the end user named in the ticket.

The infrastructure provides a central user identity service which issues these “end user permission tickets”. An end user login is verified by the central identity service which then issues a user credential, such as a cookie or OAuth token, to the user’s client device. Every subsequent request from the client device into Google needs to present that user credential.

When a service receives an end user credential, it passes the credential to the central identity service for verification. If the end user credential verifies correctly, the central identity service returns a short-lived “end user permission ticket” that can be used for RPCs related to the request. In our example, that service which gets the “end user permission ticket” would be the Gmail service, which would pass it to the Contacts service. From that point on, for any cascading calls, the “end user permission ticket” can be handed down by the calling service to the callee as a part of the RPC call.



[Figure 2]
Service Identity and Access Management

The infrastructure provides service identity, automatic mutual authentication, encrypted inter-service communication and enforcement of access policies defined by the service owner.

Secure Data Storage

Up to this point in the discussion, we have described how we deploy services securely. We now turn to discussing how we implement secure data storage on the infrastructure.

Encryption at Rest

Google's infrastructure provides a variety of storage services, such as BigTable and Spanner, and a central key management service. Most applications at Google access physical storage indirectly via these storage services. The storage services can be configured to use keys from the central key management service to encrypt data before it is written to physical storage. This key management service supports automatic key rotation, provides extensive audit logs, and integrates with the previously mentioned end user permission tickets to link keys to particular end users.

Performing encryption at the application layer allows the infrastructure to isolate itself from potential threats at the lower levels of storage such as malicious disk firmware. That said, the infrastructure also implements additional layers of protection. We enable hardware encryption support in our hard drives and SSDs and meticulously track each drive through its lifecycle. Before a decommissioned encrypted storage device can physically leave our custody, it is cleaned using a multi-step process that includes two independent verifications. Devices that do not pass this wiping procedure are physically destroyed (e.g. shredded) on-premise.

Deletion of Data

Deletion of data at Google most often starts with marking specific data as "scheduled for deletion" rather than actually removing the data entirely. This allows us to recover from unintentional deletions, whether customer-initiated or due to a bug or process error internally. After having been marked as "scheduled for deletion," the data is deleted in accordance with service-specific policies.

To protect against sophisticated adversaries who may be trying to tap our private WAN links, the infrastructure automatically encrypts all infrastructure RPC traffic which goes over the WAN between data centers.

When an end user deletes their entire account, the infrastructure notifies services handling end user data that the account has been deleted. The services can then schedule data associated with the deleted end user accountfor deletion.

Secure Internet Communication

Until this point in this document, we have described how we secure services on our infrastructure. In this section we turn to describing how we secure communication between the internet and these services.

As discussed earlier, the infrastructure consists of a large set of physical machines which are interconnected over the LAN and WAN and the security of inter-service communication is not dependent on the security of the network. However, we do isolate our infrastructure from the internet into a private IP space so that we can more easily implement additional protections such as defenses against denial of service (DoS) attacks by only exposing a subset of the machines directly to external internet traffic.

Google Front End Service

When a service wants to make itself available on the Internet, it can register itself with an infrastructure service called the Google Front End (GFE). The GFE ensures that all TLS connections are terminated using correct certificates and following best practices such as supporting perfect forward secrecy. The GFE additionally applies protections against Denial of Service attacks (which we will discuss in more detail later). The GFE then forwards requests for the service using the RPC security protocol discussed previously.

In effect, any internal service which chooses to publish itself externally uses the GFE as a smart reverse-proxy front end. This front end provides public IP hosting of its public DNS name, Denial of Service (DoS) protection, and TLS termination. Note that GFEs run on the infrastructure like any other service and thus have the ability to scale to match incoming request volumes.

The Google Front End ensures that all TLS connections are terminated using correct certificates and following best practices such as supporting perfect forward secrecy.

Denial of Service (DoS) Protection

The sheer scale of our infrastructure enables Google to simply absorb many DoS attacks. That said, we have multi-tier, multi-layer DoS protections that further reduce the risk of any DoS impact on a service running behind a GFE.

After our backbone delivers an external connection to one of our data centers, it passes through several layers of hardware and software load-balancing. These load balancers report information about incoming traffic to a central DoS service running on the infrastructure. When the central DoS service detects that a DoS attack is taking place, it can configure the load balancers to drop or throttle traffic associated with the attack.

At the next layer, the GFE instances also report information about requests that they are receiving to the central DoS service, including application layer information that the load balancers don't have. The central DoS service can then also configure the GFE instances to drop or throttle attack traffic.

User Authentication

After DoS protection, the next layer of defense comes from our central identity service. This service usually manifests to end users as the Google login page. Beyond asking for a simple username and password, the service also intelligently challenges users for additional information based on risk factors such as whether they have logged in from the same device or a similar location in the past. After authenticating the user, the identity service issues credentials such as cookies and OAuth tokens that can be used for subsequent calls.

Users also have the option of employing second factors such as OTPs or phishing-resistant Security Keys when signing in. To ensure that the benefits go beyond Google, we have worked in the FIDO Alliance with multiple device vendors to develop the Universal 2nd Factor (U2F) open standard. These devices are now available in the market and other major web services also have followed us in implementing U2F support

The sheer scale of our infrastructure enables Google to simply absorb many DoS attacks. That said, we have multi-tier, multi-layer DoS protections that further reduce the risk of any DoS impact on a service running behind a GFE.

Operational Security

Up to this point we have described how security is designed into our infrastructure and have also described some of the mechanisms for secure operation such as access controls on RPCs.

We now turn to describing how we actually operate the infrastructure securely: We create infrastructure software securely, we protect our employees' machines and credentials, and we defend against threats to the infrastructure from both insiders and external actors.

Safe Software Development

Beyond the central source control and two-party review features described earlier, we also provide libraries that prevent developers from introducing certain classes of security bugs. For example, we have libraries and frameworks that eliminate XSS vulnerabilities in web apps. We also have automated tools for automatically detecting security bugs including fuzzers, static analysis tools, and web security scanners.

As a final check, we use manual security reviews that range from quick triages for less risky features to in-depth design and implementation reviews for the most risky features. These reviews are conducted by a team that includes experts across web security, cryptography, and operating system security. The reviews can also result in new security library features and new fuzzers that can then be applied to other future products.

In addition, we run a Vulnerability Rewards Program where we pay anyone who is able to discover and inform us of bugs in our infrastructure or applications. We have paid several million dollars in rewards in this program.

Google also invests a large amount of effort in finding 0-day exploits and other security issues in all the open source software we use and upstreaming these issues. For example, the OpenSSL Heartbleed bug was found at Google and we are the largest submitter of CVEs and security bug fixes for the Linux KVM hypervisor.

We run a Vulnerability Rewards Program where we pay anyone who is able to discover and inform us of bugs in our infrastructure or applications.

Keeping Employee Devices and Credentials Safe

We make a heavy investment in protecting our employees' devices and credentials from compromise and also in monitoring activity to discover potential compromises or illicit insider activity. This is a critical part of our investment in ensuring that our infrastructure is operated safely.

Sophisticated phishing has been a persistent way to target our employees. To guard against this threat we have replaced phishable OTP second factors with mandatory use of U2F-compatible Security Keys for our employee accounts.

We make a large investment in monitoring the client devices that our employees use to operate our infrastructure. We ensure that the operating system images for these client devices are up-to-date with security patches and we control the applications that can be installed. We additionally have systems for scanning user-installed apps, downloads, browser extensions, and content browsed from the web for suitability on corp clients.

Being on the corporate LAN is not our primary mechanism for granting access privileges. We instead use application-level access management controls which allow us to expose internal applications to only specific users when they are coming from a correctly managed device and from expected networks and geographic locations. (For more detail see our additional reading about 'BeyondCorp'.)

Reducing Insider Risk

We aggressively limit and actively monitor the activities of employees who have been granted administrative access to the infrastructure and continually work to eliminate the need for privileged access for particular tasks by providing automation that can accomplish the same tasks in a safe and controlled way. This includes requiring two-party approvals for some actions and introducing limited APIs that allow debugging without exposing sensitive information.

Google employee access to end user information can be logged through low-level infrastructure hooks. Google's security team actively monitors access patterns and investigates unusual events.

Intrusion Detection

Google has sophisticated data processing pipelines which integrate host-based signals on individual devices, network-based signals from various monitoring points in the infrastructure, and signals from infrastructure services. Rules and machine intelligence built on top of these pipelines give operational security engineers warnings of possible incidents. Our investigation and incident response t eams triage, investigate, and respond to these potential incidents 24 hours a day, 365 days a year. We conduct Red Team exercises to measure and improve the effectiveness of our detection and response mechanisms.

Rules and machine intelligence built on top of signal monitoring pipelines give operational Security Engineers warnings of possible incidents.

Securing the Google Cloud Platform (GCP)

In this section, we highlight how our public cloud infrastructure, GCP, benefits from the security of the underlying infrastructure. In this section, we will take Google Compute Engine (GCE) as an example service and describe in detail the service-specific security improvements that we build on top of the infrastructure.

GCE enables customers to run their own virtual machines on Google's infrastructure. The GCE implementation consists of several logical components, most notably the management control plane and the virtual machines themselves.

The management control plane exposes the external API surface and orchestrates tasks like virtual machine creation and migration. It runs as a variety of services on the infrastructure, thus it automatically gets foundational integrity features such as a secure boot chain. The individual services run under distinct internal service accounts so that every service can be granted only the permissions it requires when making remote procedure calls (RPCs) to the rest of the control plane. As discussed earlier, the code for all of these services is stored in the central Google source code repository, and there is an audit trail between this code and the binaries that are eventually deployed.

The GCE control plane exposes its API via the GFE, and so it takes advantage of infrastructure security features like Denial of Service (DoS) protection and centrally managed SSL/TLS support. Customers can get similar protections for applications running on their GCE VMs by choosing to use the optional Google Cloud Load Balancer service which is built on top of the GFE and can mitigate many types of DoS attacks.

End user authentication to the GCE control plane API is done via Google's centralized identity service which provides security features such as hijacking detection. Authorization is done using the central Cloud IAM service.

The network traffic for the control plane, both from the GFEs to the first service behind it and between other control plane services is automatically authenticated by the infrastructure and encrypted whenever it travels from one data center to another. Additionally, the infrastructure has been configured to encrypt some of the control plane traffic within the data center as well.

Each virtual machine (VM) runs with an associated virtual machine manager (VMM) service instance. The infrastructure provides these services with two identities. One identity is used by the VMM service instance for its own calls and one identity is used for calls that the VMM makes on behalf of the customer's VM. This allows us to further segment the trust placed in calls coming from the VMM.

GCE persistent disks are encrypted at-rest using keys protected by the central infrastructure key management system. This allows for automated rotation and central auditing of access to these keys.

Customers today have the choice of whether to send traffic from their VMs to other VMs or the internet in the clear, or to implement any encryption they choose for this traffic. We have started rolling out automatic encryption for the WAN traversal hop of customer VM to VM traffic. As described earlier, all control plane WAN traffic within the infrastructure is already encrypted. In the future we plan to take advantage of the hardware-accelerated network encryption discussed earlier to also encrypt inter-VM LAN traffic within the data center.

The Google Compute Engine (GCE) control plane exposes its API via the Google Front-end (GFE), and so it takes advantage of infrastructure security features like Denial of Service (DoS) protection and centrally managed SSL/TLS support.

The isolation provided to the VMs is based on hardware virtualization using the open source KVM stack. We have further hardened our particular implementation of KVM by moving some of the control and hardware emulation stack into an unprivileged process outside the kernel. We have also extensively tested the core of KVM using techniques like fuzzing, static analysis, and manual code review. As mentioned earlier, the majority of the recently publicly disclosed vulnerabilities which have been upstreamed into KVM came from Google.

Finally, our operational security controls are a key part of making sure that accesses to data follow our policies. As part of the Google Cloud Platform, GCE's use of customer data follows the GCP use of customer data policy, namely that Google will not access or use customer data, except as necessary to provide services to customers.

Conclusion

We have described how the Google infrastructure is designed to build, deploy and operate services securely at internet scale. This includes both consumer services such as Gmail and our enterprise services. In addition, our Google Cloud offerings are built on top of this same infrastructure.

We invest heavily in securing our infrastructure. We have many hundreds of engineers dedicated to security and privacy distributed across all of Google, including many who are recognized industry authorities.

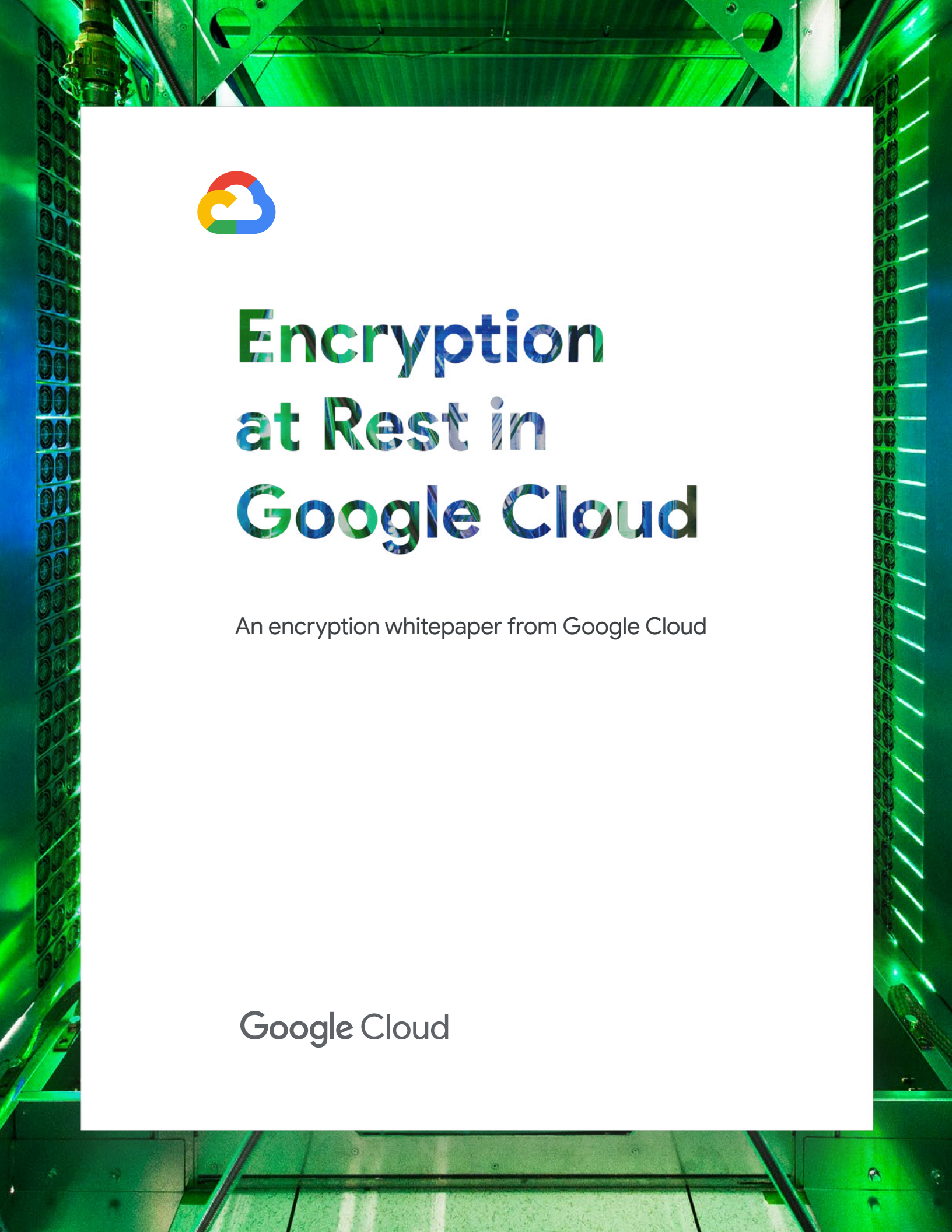
As we have seen, the security in the infrastructure is designed in layers starting from the physical components and data center, to hardware provenance, and then on to secure boot, secure inter-service communication, secured data at rest, protected access to services from the internet and finally, the technologies and people processes we deploy for operational security.

We invest heavily in securing our infrastructure. We have many hundreds of engineers dedicated to security & privacy distributed across all of Google, including many who are recognized industry authorities.

Additional Reading

Please see the following papers for more detail on specific areas:

1. Physical security of our data centers
   https://goo.gl/WYIKGG
2. Design of our cluster management and orchestration
   http://research.google.com/pubs/pub43438.html
3. Storage encryption and our customer facing GCP encryption features
   https://cloud.google.com/security/encryption-at-rest/
4. BigTable storage service
   http://research.google.com/archive/bigtable.html
5. Spanner storage service
   http://research.google.com/archive/spanner.html
6. Architecture of our network load balancing
   http://research.google.com/pubs/pub44824.html
7. BeyondCorp approach to enterprise security
   http://research.google.com/pubs/pub43231.html
8. Combating phishing with Security Key & the Universal 2nd Factor (U2F) standard
   http://research.google.com/pubs/pub45409.html
9. More about the Google Vulnerability Rewards Program
   https://bughunter.withgoogle.com/
10. More about HTTPs and other load balancing offerings on GCP
    https://cloud.google.com/compute/docs/load-balancing/
11. More about DoS protection best practices on GCP
    https://cloud.google.com/files/GCPDDoSprotection-04122016.pdf
12. Google Cloud Platform use of customer data policy
    https://cloud.google.com/terms/
13. More about application security & compliance in G Suite (Gmail, Drive, etc)
    https://goo.gl/3J20R2



Encryption at Rest in Google Cloud

An encryption whitepaper from Google Cloud

Google Cloud

Google Cloud

Table of Contents

This is the second of two whitepapers on how Google uses encryption to protect your data. We also released a G Suite encryption whitepaper. You may still find it useful to read both documents to learn about the use of encryption at Google.

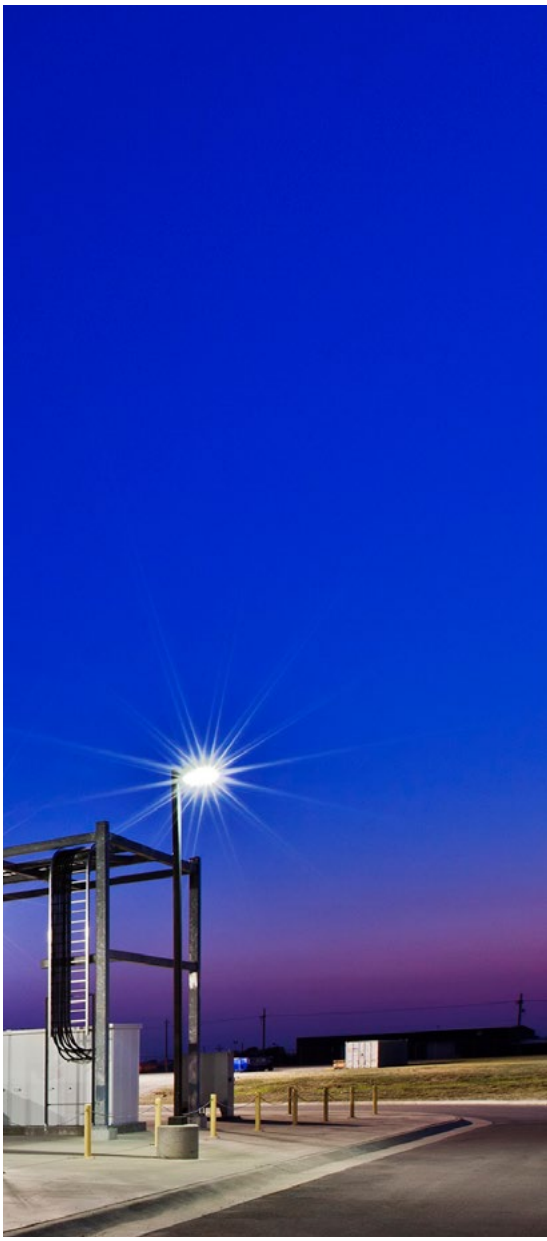
In this whitepaper, you will find more detail on Google's key hierarchy and root of trust, as well as information on the granularity of encryption in specific GCP services for data at rest (this document does not cover encryption in transit).

For all Google products, we strive to keep customer data highly protected, and to be as transparent as possible about how we secure it.

The content contained herein is correct as of August 2016, and represents the status quo as of the time it was written. Google Cloud Platform's security policies and systems may change going forward, as we continually improve protection for our customers.

CIO-level summary

- Google uses several layers of encryption to protect customer data at rest in Google Cloud Platform products.
- Google Cloud Platform encrypts customer content stored at rest, without any action required from the customer, using one or more encryption mechanisms. There are some minor exceptions, noted further in this document.
- Data for storage is split into chunks, and each chunk is encrypted with a unique data encryption key. These data encryption keys are stored with the data, encrypted with ("wrapped" by) key encryption keys that are exclusively stored and used inside Google's central Key Management Service. Google's Key Management Service is redundant and globally distributed.
- Data stored in Google Cloud Platform is encrypted at the storage level using either AES256 or AES128.
- Google uses a common cryptographic library, Keyczar, to implement encryption consistently across almost all Google Cloud Platform products. Because this common library is widely accessible, only a small team of cryptographers needs to properly implement and maintain this tightly controlled and reviewed code.



Introduction

For many individuals and companies, security is a deciding factor in choosing a public cloud vendor. At Google, security is of the utmost importance. We take security and privacy seriously, and we work tirelessly to protect your data — whether it is traveling over the Internet, moving between our data centers, or stored on our servers.

Central to our comprehensive security strategy is encryption in transit and at rest, which ensures the data can be accessed only by the authorized roles and services with audited access to the encryption keys. This paper describes Google's approach to encryption at rest for the Google Cloud Platform, and how Google uses it to keep your information more secure.

This document is targeted at CISOs and security operations teams currently using or considering using Google Cloud Platform. With the exception of the introduction, this document assumes a basic understanding of encryption and cryptographic primitives.

What is encryption?

Encryption is a process that takes legible data as input (often called plaintext), and transforms it into an output (often called ciphertext) that reveals little or no information about the plaintext. The encryption algorithm used is public, such as the Advanced Encryption Standard (AES), but execution depends on a key, which is kept secret. To decrypt the ciphertext back to its original form, you need to employ the key. At Google, the use of encryption to keep data confidential is usually combined with integrity protection; someone with access to the ciphertext can neither understand it nor make a modification without knowledge of the key. For more information on cryptography, a good resource is an Introduction to Modern Cryptography.

Encryption is a process that takes legible data as input, called plaintext, and transforms it into an output, called ciphertext, that reveals little or no information about the plaintext.

In this whitepaper, we focus on encryption at rest. By encryption at rest, we mean encryption used to protect data that is stored on a disk (including solid-state drives) or backup media.

Why encryption helps secure customer data

Encryption is one piece of a broader security strategy. Encryption adds a layer of defense in depth for protecting data — encryption ensures that if the data accidentally falls into an attacker's hands, they cannot access the data without also having access to the encryption keys. Even if an attacker obtains the storage devices containing your data, they won't be able to understand or decrypt it.

Encryption at rest reduces the surface of attack by effectively "cutting out" the lower layers of the hardware and software stack. Even if these lower layers are compromised (for example, through physical access to devices), the data on those devices is not compromised if adequate encryption is deployed. Encryption also acts as a "choke-point" — centrally managed encryption keys create a single place where access to data is enforced and can be audited.

Encryption provides an important mechanism in how Google ensures the privacy of customer data — it allows systems to manipulate data, e.g., for backup, and engineers to support our infrastructure, without providing access to content..

What we consider customer data

As defined in the Google Cloud Platform terms of service, customer data refers to content provided to Google by a Google Cloud Platform customer (or at their direction), directly or indirectly, via Google Cloud Platform services used bythat customer's account. Customer data includes customer content and customer metadata.

Customer content is data that Google Cloud Platform customers generate themselves or provide to Google, like data stored in Google Cloud Storage, disk snapshots used by Google Compute Engine, and Cloud IAM policies. The encryption at rest of customer content is the focus of this whitepaper.

Customer metadata makes up the rest of customer data, and refers to all data that cannot be classified as customer content. This could include auto-generated project numbers, timestamps, and IP addresses, as well as the byte size of an object in Google Cloud Storage, or the machine type in Google Compute Engine. Metadata is protected to a degree that is reasonable for ongoing performance and operations.

Customer data refers to content provided to Google by a Google Cloud Platform customer or at their direction, directly or indirectly, via Cloud services used by that customer's account.

Google's default encryption

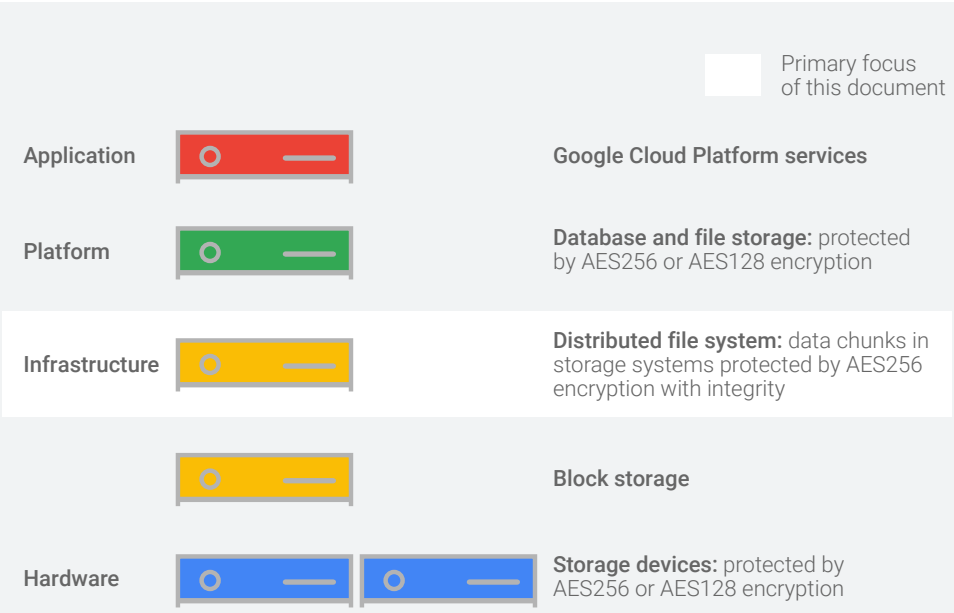
Encryption of data at rest

Layers of encryption

Google uses several layers of encryption to protect data. Using multiple layers of encryption adds redundant data protection and allows us to select the optimal approach based on application requirements.

Encryption at the storage system layer

To understand how specifically Google Cloud Storage encryption works, it's important to understand how Google stores customer data. Data is broken into subfile *chunks* for storage; each chunk can be up to several GB in size. Each chunk is encrypted at the storage level with an individual encryption key: two chunks will not have the same encryption key, even if they are part of the same Google Cloud



[Figure 1]

Several layers of encryption are used to protect data stored in Google Cloud Platform. Either distributed file system encryption or database and file storage encryption is in place for almost all files; and storage device encryption is in place for almost all files.

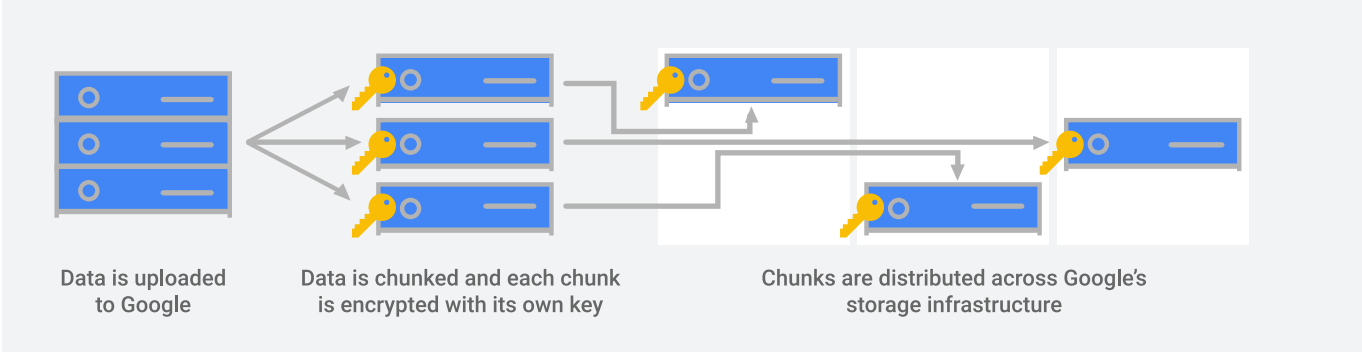
Storage object, owned by the same customer, or stored on the same machine[1]. If a chunk of data is updated, it is encrypted with a new key, rather than by reusing the existing key. This partition of data, each using a different key, means the "blast radius" of a potential data encryption key compromise is limited to only that data chunk.

Google encrypts data prior to it being written to disk. Encryption is inherent in all of Google's storage systems — rather than added on afterward.

Each data chunk has a unique identifier. Access control lists (ACLs) ensure that each chunk can be decrypted only by Google services operating under authorized roles, which are granted access at that point in time. This prevents access to the data without authorization, bolstering both data security and privacy.

Each chunk is distributed across Google's storage systems, and is replicated in encrypted form for backup and disaster recovery. A malicious individual who wanted to access customer data would need to know and be able to access (1) all storage chunks corresponding to the data they want, and (2) the encryption keys corresponding to the chunks.

Each data chunk is encrypted at the storage level with an individual encryption key — two chunks will not have the same encryption key, even if they are part of the same Google Cloud Storage object, owned by the same customer, or stored on the same machine.



[Figure 2]

Data at Google is broken up into encrypted chunks for storage.

Google uses the Advanced Encryption Standard (AES) algorithm to encrypt data at rest. AES is widely used because (1) both AES256 and AES128 are recommended by the National Institute of Standards and Technology (NIST) for long-term storage use (as of November 2015), and (2) AES is often included as part of customer compliance requirements.

[1] Data chunks in Cloud Datastore, App Engine, and Cloud Pub/Sub may contain two customers' data. See the section on granularity of data encryption by service

Data stored across Google Cloud Storage is encrypted at the storage level using AES, in Galois/Counter Mode (GCM) in almost all cases. This is implemented in the BoringSSL library that Google maintains. This library was forked from OpenSSL for internal use, after many flaws were exposed in OpenSSL. In select cases, AES is used in Cipher Block Chaining (CBC) mode with a hashed message authentication code (HMAC) for authentication; and for some replicated files, AES is used in Counter (CTR) mode with HMAC. (Further details on algorithms are provided later in this document.) In other Google Cloud Platform products, AES is used in a variety of modes.

Encryption at the storage device layer

In addition to the storage system level encryption described above, in most cases data is also encrypted at the storage device level, with at least AES128 for hard disks (HDD) and AES256 for new solid state drives (SSD), using a separate device-level key (which is different than the key used to encrypt the data at the storage level). As older devices are replaced, solely AES256 will be used for device-level encryption.

Encryption of backup

Google's backup system ensures that data remains encrypted throughout the backup process. This approach avoids unnecessarily exposing plaintext data.

In addition, the backup system further encrypts each backup file independently with its own data encryption key (DEK), derived from a key stored in Google's Key Management Service (KMS) plus a randomly generated per-file seed at backup time. Another DEK is used for all metadata in backups, which is also stored in Google's KMS. (Further information on key management is in a later section.)

Are there cases where data is *not* encrypted at rest?

Google Cloud Platform encrypts customer content stored at rest, without any action from the customer, using one or more encryption mechanisms, with the following exceptions.

- Serial console logs from virtual machines in Google Compute Engine; this is currently being remediated

Google Cloud Platform encrypts customer content stored at rest, without any action from the customer, using one or more encryption mechanisms, except for some minor exceptions.

- Core dumps written to local drives, when a process fails unexpectedly; this is currently being remediated
- Debugging logs written to local disk; this is currently being remediated
- Temporary files used by storage systems; this is currently being remediated
- Some logs that may include customer content as well as customer metadata; this is planned for remediation

This data is still protected extensively by the rest of Google's security infrastructure, and in almost all cases still protected by storage-level encryption.

Key Management

Data encryption keys, key encryption keys, and Google's Key Management Service

The key used to encrypt the data in a chunk is called a *data encryption key (DEK)*. Because of the high volume of keys at Google, and the need for low latency and high availability, these keys are stored near the data that they encrypt. The DEKs are encrypted with (or "wrapped" by) a *key encryption key (KEK)*. One or more KEKs exist for each Google Cloud Platform service. These KEKs are stored centrally in *Google's Key Management Service (KMS)*, a repository built specifically for storing keys. Having a smaller number of KEKs than DEKs and using a central key management service makes storing and encrypting data at Google scale manageable, and allows us to track and control data access from a central point.

The key used to encrypt the data in a chunk is called a data encryption key (DEK). The DEKs are encrypted with a key encryption key (KEK). KEKs are stored centrally in Google's Key Management Service (KMS), a repository built specifically for this purpose.

For each Google Cloud Platform customer, any non-shared resources[2] are split into data chunks and encrypted with keys separate from keys used for other customers[3]. These DEKs are even separate from those that protect other pieces of the same data owned by that same customer.

[2] An example of a shared resource (where this segregation does not apply) would be a shared base image in Google Compute Engine — naturally, multiple customers refer to a single copy, which is encrypted by a single DEK.
[3] With the exception of data stored in Cloud Datastore, App Engine, and Cloud Pub/Sub, where two customers' data may be encrypted with the same DEK. See the section on granularity of data encryption by service.

DEKs are generated by the storage system using Google's common cryptographic library. They are then sent to KMS to wrap with that storage system's KEK, and the wrapped DEKs are passed back to the storage system to be kept with the data chunks. When a storage system needs to retrieve encrypted data, it retrieves the wrapped DEK and passes it to KMS. KMS then verifies that this service is authorized to use the KEK, and if so, unwraps and returns the plaintext DEK to the service. The service then uses the DEK to decrypt the data chunk into plaintext and verify its integrity.

Most KEKs for encrypting data chunks are generated within KMS, and the rest are generated inside the storage services. For consistency, all KEKs are generated using Google's common cryptographic library, using a random number generator (RNG) built by Google. This RNG is based on NIST 800-90A and generates an AES256 KEK[4]. This RNG is seeded from the Linux kernel's RNG, which in turn is seeded from multiple independent entropy sources. This includes entropic events from the data center environment, such as fine-grained measurements of disk seeks and inter-packet arrival times, and Intel's RDRAND instruction where it is available (on Ivy Bridge and newer CPUs).

Data stored in Google Cloud Platform is encrypted with DEKs using AES256 orAES128, as described above; and any new data encrypted in persistent disks in Google Compute Engine is encrypted using AES256. DEKs are wrapped with KEKs using AES256 or AES128, depending on the Google Cloud Platform service. We are currently working on upgrading all KEKs for Cloud services to AES256.

Google's KMS manages KEKs, and was built solely for this purpose. It was designed with security in mind. KEKs are not exportable from Google's KMS by design; all encryption and decryption with these

keys must be done within KMS. This helps prevent leaks and misuse, and enables KMS to emit an audit trail when keys are used.

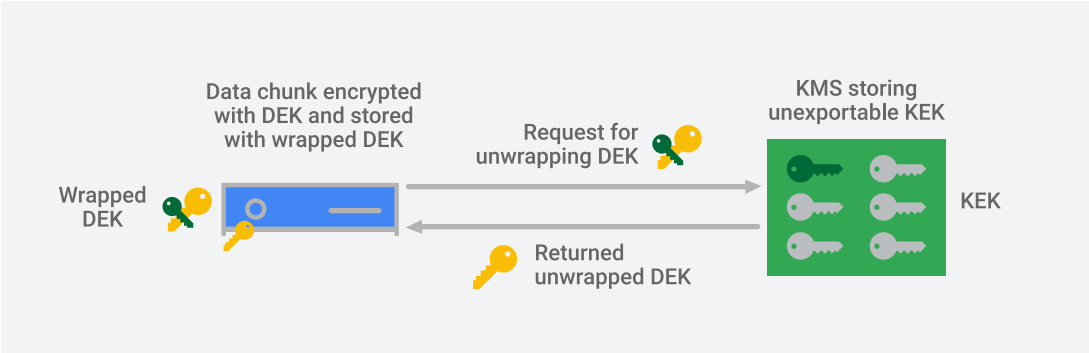
KMS can automatically rotate KEKs at regular time intervals, using Google's common cryptographic library to generate new keys. Though we often refer to just a single key, we really mean that data is protected using a *key set*: one key active for encryption and a set of

KEKs are not exportable from Google's KMS by design — all encryption and decryption with these keys must be done within KMS. This helps prevent leaks and misuse.

[4] Note that in the past, this was AES128, and some of these keys remain active for decrypting data.

historical keys for decryption, the number of which is determined by the key *rotation schedule*. The actual rotation schedule for a KEK varies by service, but the standard rotation period is 90 days. Google Cloud Storage specifically rotates its KEKs every 90 days, and can store up to 20 versions, requiring re-encryption of data at least once every 5 years (though in practice, data re-encryption is much more frequent). KMS-held keys are backed up for disaster recovery purposes, and they are indefinitely recoverable.

The use of KEKs is managed by access control lists (ACLs) in KMS for each key,with a per-key policy. Only authorized Google services and users are allowed access to a key. The use of each key is tracked at the level of the individual operation that requires that key — so every time an individual uses a key, it is authenticated and logged. All human data accesses are auditable as part of Google's overall security and privacy policies.



[Figure 3]

To decrypt a data chunk, the storage service calls Google's Key Management Service (KMS) to retrieve the unwrapped data encryption key (DEK) for that data chunk.

When a Google Cloud Platform service accesses an encrypted chunk of data, here's what happens:

1. The service makes a call to the storage system for the data it needs.

2. The storage system identifies the chunks in which that data is stored (the chunk IDs) and where they are stored.

3. For each chunk, the storage system pulls the wrapped DEK stored with that chunk (in some cases, this is done by the service), and sends it to KMS for unwrapping.

4. The storage system verifies that the identified job is allowed to access that data chunk based on a job identifier, and using the chunk ID; and KMS verifies that the storage system is authorized both to use the KEK associated with the service, and to unwrap that specific DEK.

5. KMS does one of the following:
   - Passes the unwrapped DEK back to the storage system, which decrypts the data chunk and passes it to the service. Or, in some rare cases,
   - Passes the unwrapped DEK to the service; the storage system passes the encrypted data chunk to the service, which decrypts the data chunk and uses it.

This process is different in dedicated storage devices, such as local SSDs, wherethe device manages and protects the device-level DEK.

Encryption key hierarchy and root of trust

Google's KMS is protected by a root key called the *KMS master key*, which wraps all the KEKs in KMS. This KMS master key is AES256[5], and is itself stored in another key management service, called the *Root KMS*. Root KMS stores a much smaller number of keys—approximately a dozen. For additional security, Root KMS is not run on general production machines, but instead is run only on dedicated machines in each Google data center.

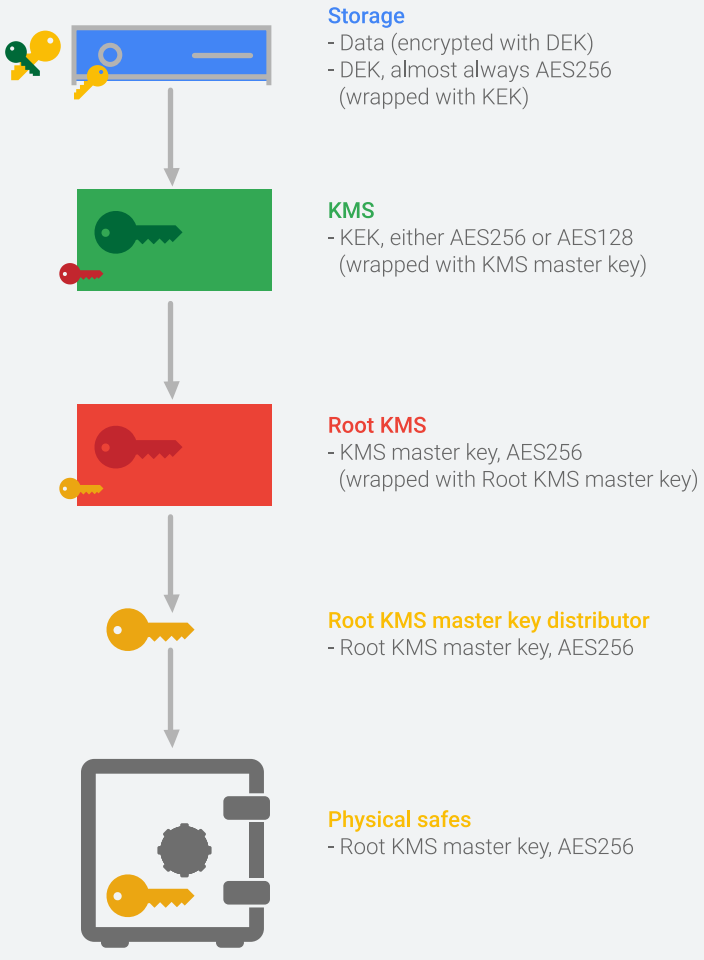
Root KMS in turn has its own root key, called the *root KMS master key*, which is also AES256[6] and is stored in a peer-to-peer infrastructure, the *root KMS master key distributor*, which replicates these keys globally. The root KMS master key distributor only holds the keys in RAM on the same dedicated machines as Root KMS, and uses logging to verify proper use. One instance of the root KMS master key distributor runs for every instance of Root KMS. (The root KMS master key distributor is still being phased in, to replace a system that operated in a similar manner but was not peer to peer.)

Google's root of trust, the root KMS master key, is kept in RAM and is also secured in physical safes in limited Google locations in case of a global restart.

[5] Note that in the past, this was AES128, and some of these keys remain active for decrypting data.
[6] Note that in the past, this was AES128, and some of these keys remain active for decrypting data.

When a new instance of the root KMS master key distributor is started, it is configured with a list of host names already running distributor instances. Distributor instances can then obtain the root KMS master key from other running instances. Other than the disaster-recovery mechanisms described below, the root KMS master key exists only in RAM on a limited number of specially secured machines.

To address the scenario where all instances of the root KMS master key distributor restart simultaneously, the root KMS master key is also backed up on secure hardware devices stored in physical safes in highly secured areas in two physically separated, global Google locations. This backup would be needed only if all distributor instances were to go down at once; for example, in a global restart. Fewer than 20 Google employees are able to access these safes.



[Figure 4]

The encryption key hierarchy protects a chunk of data with a DEK, wrapped with a KEK in KMS, which is in turn protected by Root KMS and the root KMS master key distributor.

To summarize:

- Data is chunked and encrypted with DEKs
- DEKs are encrypted with KEKs
- KEKs are stored in KMS
- KMS is run on multiple machines in data centers globally
    - KMS keys are wrapped with the KMS master key, which is stored in Root KMS
- Root KMS is much smaller than KMS and runs only on dedicated machines in each data center
    - Root KMS keys are wrapped with the root KMS master key, which is stored in the root KMS master key distributor
- The root KMS master key distributor is a peer-to-peer infrastructure running concurrently in RAM globally on dedicated machines; each gets its key material from other running instances
    - If all instances of the distributor were to go down (total shutdown), a master key is stored in (different) secure hardware in (physical) safes in limited Google locations.
- The root KMS master key distributor is currently being phased in, to replace a system that operated in a similar manner but was not peer to peer.

Global availability and replication

High availability and low latency, global access to keys, are critical at every level; these characteristics are needed for key management services to be used across Google.

For this reason, KMS is highly scalable, and it is replicated thousands of timesin Google's data centers globally. It is run on regular machines in Google's production fleet, and instances of KMS run globally to support Google Cloud Platform operations. As a result, the latency of any single key operation is very low.

Root KMS is run on several machines dedicated to security operations, in each data center. The root KMS master key distributor is run on these same machines, one-to-one with Root KMS. The root KMS master key distributor provides a distribution mechanism via a gossiping protocol — at a fixed time interval, each instance of the

distributor picks a random other instance to compare its keys with, and reconciles any differences in key versions. With this model, there is no central node that all of Google's infrastructure depends on; this allows Google to maintain and protect key material with high availability.

Google's common cryptographic library

Google's common cryptographic library is *Keyczar*[7], which implements cryptographic algorithms using BoringSSL[8]. Keyczar is available to all Google developers. Because this common library is widely accessible, only a small team of cryptographers needs to properly implement this tightly controlled and reviewed code — it's not necessary for every team at Google to "roll their own" cryptography. A special Google security team is responsible for maintaining this common cryptographic library for all products.

The Keyczar encryption library supports a wide variety of encryption key types and modes, and these are reviewed regularly to ensure they are current with the latest attacks.

At the time of this document's publication, Google uses the following encryption algorithms for encryption at rest for DEKs and KEKs. These are subject to change as we continue to improve our capabilities and security.

> Google uses a widely accessible common cryptographic library that is tightly managed, controlled and reviewed by a small team of cryptographers — so that it is not necessary for every team at Google to "roll their own" cryptography.

Cryptographic primitive	Preferred protocols	Other supported protocols[9]
Symmetric encryption	• AES-GCM (256 bits)	• AES-CBC and AES-CTR (128 and 256 bits) • AES-EAX (128 and 256 bits)
Symmetric signatures (where used with AES-CBC and AES-CTR above for authentication)	• HMAC-SHA256	• HMAC-SHA512 • HMAC-SHA1

[7] An older version of Keyczar has been open-sourced, but the open-source version has not been updated recently and does not reflect internal developments.
[8] OpenSSL is also in use, in some places in Google Cloud Storage.
[9] Other cryptographic protocols exist in the library and were historically supported, but this list covers the primary uses in Google Cloud Platform.

Granularity of encryption in each Google Cloud Platform product

Each Google Cloud Platform service splits data at a different level of granularity for encryption.

		Granularity of customer data encryption[10] (size of data encrypted with a single DEK)
Storage	Cloud Storage	Per data chunk (typically 256KB-8MB)
	Cloud SQL	• Second generation: Per instance, as in Google Compute Engine (each instance could contain multiple databases) • First generation: Per instance
	Cloud Database	Per data chunk[11]
	Cloud Bigtable	Per data chunk (several per table)
Compute	Compute Engine	• Several per disk • Per snapshot group, with individual snapshot ranges derived from the snapshot group master key • Per image
	App Engine[12]	Per data chunk[13]
	Container Engine	Several per disk, for persistent disks
	Container Registry	Stored in Google Cloud Storage, per data chunk
Big Data	BigQuery	Several per dataset
	Cloud Dataflow	Stored in Google Cloud Storage, per data chunk
	Cloud Dataproc	Stored in Google Cloud Storage, per data chunk
	Cloud Datalab	Stored in Cloud Bigtable, per notebook file
	Cloud Pub/Sub	Per one hour, for up to 1,000,000 messages[14]

[10] Refers to granularity of encryption for customer content. This does not include customer metadata, such as resource names. In some services, all metadata is encrypted with a single DEK.
[11] Not unique to a single customer.
[12] Includes application code and application settings. Data used in App Engine is stored in Cloud Datastore, Cloud SQL or Cloud Storage depending on customer configurations.
[13] Not unique to a single customer.
[14] Cloud Pub/Sub rotates the DEK used to encrypt messages every hour, or sooner if 1,000,000 messages are encrypted. Not unique to a single customer.

Additional encryption options for Cloud customers

In addition to providing encryption by default in Google Cloud Platform, we are working to offer customers additional encryption and key management options for greater control.

We want to enable Google Cloud Platform customers to:

- Remain the ultimate custodian of their data, and be able to control access to and use of that data at the finest level of granularity, to ensure both data security and privacy
- Manage encryption for their cloud-hosted data in the same way they currently do on-premises — or, ideally, better
- Have a provable and auditable root of trust over their resources
- Be able to further separate and segregate their data, beyond the use of ACLs

Customers can use existing encryption keys that they manage with the Google Cloud Platform, using the Customer supplied encryption keys feature. This feature is available for Google Cloud Storage and for Google Compute Engine.

We are currently working to introduce new encryption options. Details will be provided as they become available.

We are working to offer Google Cloud Platform customers additional encryption and key management options.

Research and innovation in cryptography

To keep pace with the evolution of encryption, Google has a team of world-class security engineers tasked with following, developing, and improving encryption technology. Our engineers take part in standard-ization processes and in maintaining widely used encryption

software. We regularly publish our research in the field of encryption so that everyone in the industry — including the general public — can benefit from our knowledge. For example, in 2014 we revealed a significant vulnerability in SSL 3.0 encryption (known as POODLE), and in 2015 we identified a high-risk vulnerability in OpenSSL.

Google plans to remain the industry leader in encryption for cloud services. In terms of developing, implementing, and researching newer cryptographic techniques, we have teams working on:

- *Partially homomorphic cryptography,* which allows some operations to be performed on data while it is encrypted, so the cloud never sees the data in plaintext, even in memory. One place this technology is being used is as part of our experimental encrypted BigQuery client, which is openly available.

- *Format- and order- preserving cryptography*, which allows some comparison and ranking operations to be performed on data while it is encrypted.

- *Post-quantum cryptography*, which allows us to replace existing crypto primitives that are vulnerable to efficient quantum attacks with post-quantum candidates that are believed to be more robust against such attacks. The primary focus here is in researching and prototyping lattice-based public-key cryptography, including NIST recommendations on post-quantum algorithms. Lattice-based crypto is currently thought to be one of the most likely encryption techniques to be used in a post-quantum world, although we are still in early days in terms of best algorithms, concrete parameters, and cryptanalysis for applying lattice-based crypto. Although symmetric key encryption and MACs are weakened by known quantum algorithms, they can still be upgraded to similar bits of security in a post-quantum world by doubling key sizes.

Google has a team of world-class security engineers tasked with following, developing, and improving encryption technology.

Further references

Google Cloud Platform security

For general information on Google Cloud Platform security, see the Security section of the Google Cloud Platform website.

Google Cloud Platform compliance

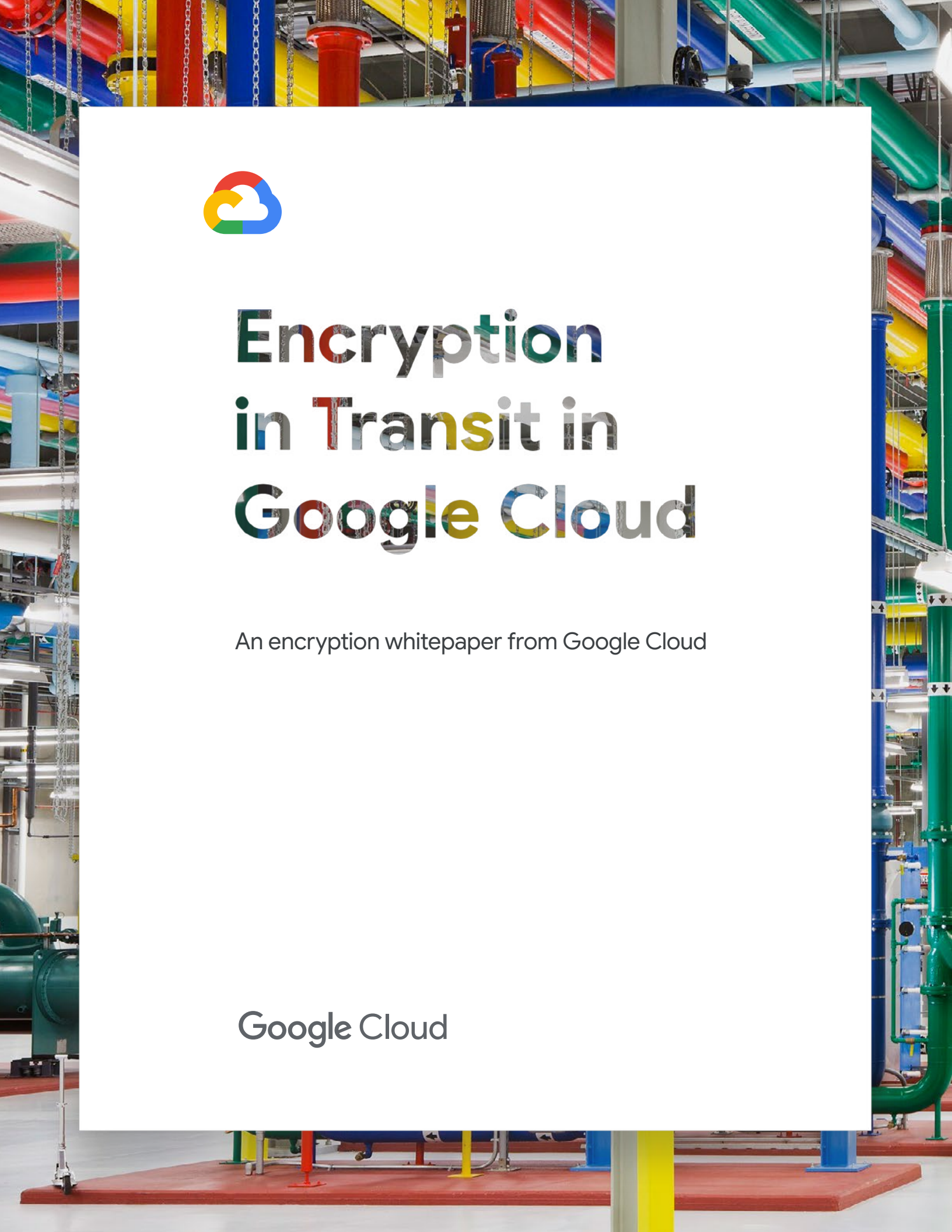
For information on Google Cloud Platform compliance and compliance certifications, see the Compliance section of the Google Cloud Platform website, which includes Google's public SOC3 audit report.

G Suite security

For information on G Suite encryption and key management, see the G Suite encryption whitepaper. That whitepaper covers much of the same content included here, but focuses solely on G Suite. For all Google Cloud solutions, we strive to keep customer data protected, and to be as transparent as possible about how we secure it.

Further information on general G Suite security is available in the Google Cloud Security and Compliance whitepaper.

We are working to offer Google Cloud Platform customers additional encryption and key management options.



Encryption in Transit in Google Cloud

An encryption whitepaper from Google Cloud

Google Cloud

Google Cloud

Table of Contents

Google Cloud

This is the third whitepaper on how Google uses encryption to protect your data. We also released Encryption at Rest in Google Cloud Platform, and G Suite encryption. You might find it useful to read these other documents to learn about the use of encryption at Google. In this whitepaper, you will find more detail on encryption in transit for Google Cloud, including Google Cloud Platform and G Suite.

For all Google products, we strive to keep customer data highly protected and to be as transparent as possible about how we secure it.

The content contained herein is correct as of November 2017. This whitepaper represents the status quo as of the time it was written. Google Cloud's security policies and systems might change going forward, as we continually improve protection for our customers.

CIO-level summary

- Google employs several security measures to help ensure the authenticity, integrity, and privacy of data in transit.
- Google encrypts and authenticates all data in transit at one or more network layers when data moves outside physical boundaries not controlled by Google or on behalf of Google. Data in transit inside a physical boundary controlled by or on behalf of Google is generally authenticated but not necessarily encrypted.
- Depending on the connection that is being made, Google applies default protections to data in transit. For example, we secure communications between the user and the Google Front End (GFE) using TLS.
- Google Cloud customers with additional requirements for encryption of data over WAN can choose to implement further protections for data as it moves from a user to an application, or virtual machine to virtual machine. These protections include IPsec tunnels, Gmail S/MIME, managed SSL certificates, and Istio.
- Google works actively with the industry to help bring encryption in transit to everyone, everywhere. We have several open-source projects that encourage the use of encryption in transit and data security on the Internet at large including Certificate Transparency, Chrome APIs, and secure SMTP.
- Google plans to remain the industry leader in encryption in transit. To this end, we dedicate resources toward the development and improvement of encryption technology. Our work in this area includes innovations in the areas of Key Transparency and post-quantum cryptography.

1. Introduction

Security is often a deciding factor when choosing a public cloud provider. At Google, security is of the utmost importance. We work tirelessly to protect your data—whether it is traveling over the Internet, moving within Google's infrastructure, or stored on our servers.

Central to Google's security strategy are authentication, integrity, and encryption, for both data at rest and in transit. This paper describes our approach to encryption in transit for Google Cloud.

For data at rest, see Encryption at Rest in Google Cloud Platform. For an overview across all of Google Security, see Google Infrastructure Security Design Overview.

Audience: this document is aimed at CISOs and security operations teams using or considering Google Cloud.

Prerequisites: in addition to this introduction, we assume a basic understanding of encryption and cryptographic primitives.

1.1 Authentication, Integrity, and Encryption

Google employs several security measures to help ensure the authenticity, integrity, and privacy of data in transit.

- **Authentication:** we verify the data source, either a human or a process, and destination.
- **Integrity:** we make sure data you send arrives at its destination unaltered.
- **Encryption:** we make your data unintelligible while in transit to keep it private.

In this paper, we focus on encryption in Google Cloud, and how we use it to protect your data. Encryption is the process through which legible data (plaintext) is made illegible (ciphertext) with the goal of ensuring the plaintext is only accessible by parties authorized by the

Encryption in transit protects your data if communications are intercepted while data moves between your site and the cloud provider or between two services.

owner of the data. The algorithms used in the encryption process are public, but the key required for decrypting the ciphertext is private. Encryption in transit often uses asymmetric key exchange, such as elliptic-curve-based Diffie-Hellman, to establish a shared symmetric key that is used for data encryption. For more information on encryption, see Introduction to Modern Cryptography.

Encryption can be used to protect data in three states:

- **Encryption at rest** protects your data from a system compromise or data exfiltration by encrypting data while stored. The Advanced Encryption Standard (AES) is often used to encrypt data at rest.

- **Encryption in transit** protects your data if communications are intercepted while data moves between your site and the cloud provider or between two services. This protection is achieved by encrypting the data before transmission; authenticating the endpoints; and decrypting and verifying the data on arrival. For example, Transport Layer Security (TLS) is often used to encrypt data in transit for transport security, and Secure/Multipurpose Internet Mail Extensions (S/MIME) is used often for email message security.

- **Encryption in use** protects your data when it is being used by servers to run computations, e.g. homomorphic encryption.

Encryption is one component of a broader security strategy. Encryption in transit defends your data, after a connection is established and authenticated, against potential attackers by:

- Removing the need to trust the lower layers of the network which are commonly provided by third parties

- Reducing the potential attack surface

- Preventing attackers from accessing data if communications are intercepted

With adequate authentication, integrity, and encryption, data that travels between users, devices, or processes can be protected in a hostile environment. The remainder of this paper explains Google's approach to the encryption of data in transit and where it is applied.

2. Google's Network Infrastructure

2.1 Physical boundaries of Google's network

Google applies different protections to data in transit when it is transmitted outside a physical boundary controlled by or on behalf of Google. A physical boundary is the barrier to a physical space that is controlled by or on behalf of Google, where we can ensure that rigorous security measures are in place. Physical access to these locations is restricted and heavily monitored. Only a small percentage of Google employees have access to hardware. Data in transit within these physical boundaries is generally authenticated, but may not be encrypted by default - you can choose which additional security measures to apply based on your threat model.

Due to the scale of the global Internet, we cannot put these same physical security controls in place for the fiber links in our WAN, or anywhere outside of physical boundaries controlled by or on behalf of Google. For this reason, we automatically enforce additional protections outside of our physical trust boundary. These protections include encryption of data in transit.

2.2 How traffic gets routed

The previous section discussed the physical boundary of Google's network and how we apply different protections to data sent outside this boundary. To fully understand how encryption in transit works at Google, it is also necessary to explain how traffic gets routed through the Internet. This section describes how requests get from an end user to the appropriate Google Cloud service or customer application, and how traffic is routed between services.

A **Google Cloud service** is a modular cloud service that we offer to our customers. These services include computing, data storage, data analytics and machine learning. For example, Google Cloud Storage

Google applies different protections to data in transit when it is transmitted outside a physical boundary controlled by or on behalf of Google. A physical boundary is the barrier to a physical space that is controlled by or on behalf of Google, where we can ensure that rigorous security measures are in place.

and Gmail are both Google Cloud services. A **customer application** is an application hosted on Google Cloud that you, as a Google customer, can build and deploy using Google Cloud services. Customer applications or partner solutions that are hosted on Google Cloud are not considered Google Cloud services[1]. For example, an application you build using Google App Engine, Google Container Engine, or a VM in Google Compute Engine is a customer application.

The five kinds of routing requests discussed below are shown in Figure 1. This figure shows the interactions between the various network components and the security in place for each connection.

End user (Internet) to a Google Cloud Service

Google Cloud services accept requests from around the world using a globally distributed system called the Google Front End (GFE). GFE terminates traffic for incoming HTTP(S), TCP and TLS proxy traffic, provides DDoS attack countermeasures, and routes and load balances traffic to the Google Cloud services themselves. There are GFE points of presence around the globe with routes advertised via unicast or Anycast.

Google Cloud services accept requests from around the world using a globally distributed system called the Google Front End (GFE). GFE terminates traffic for incoming HTTP(S), TCP and TLS proxy traffic, provides DDoS attack countermeasures, and routes and load balances traffic to the Google Cloud services themselves.

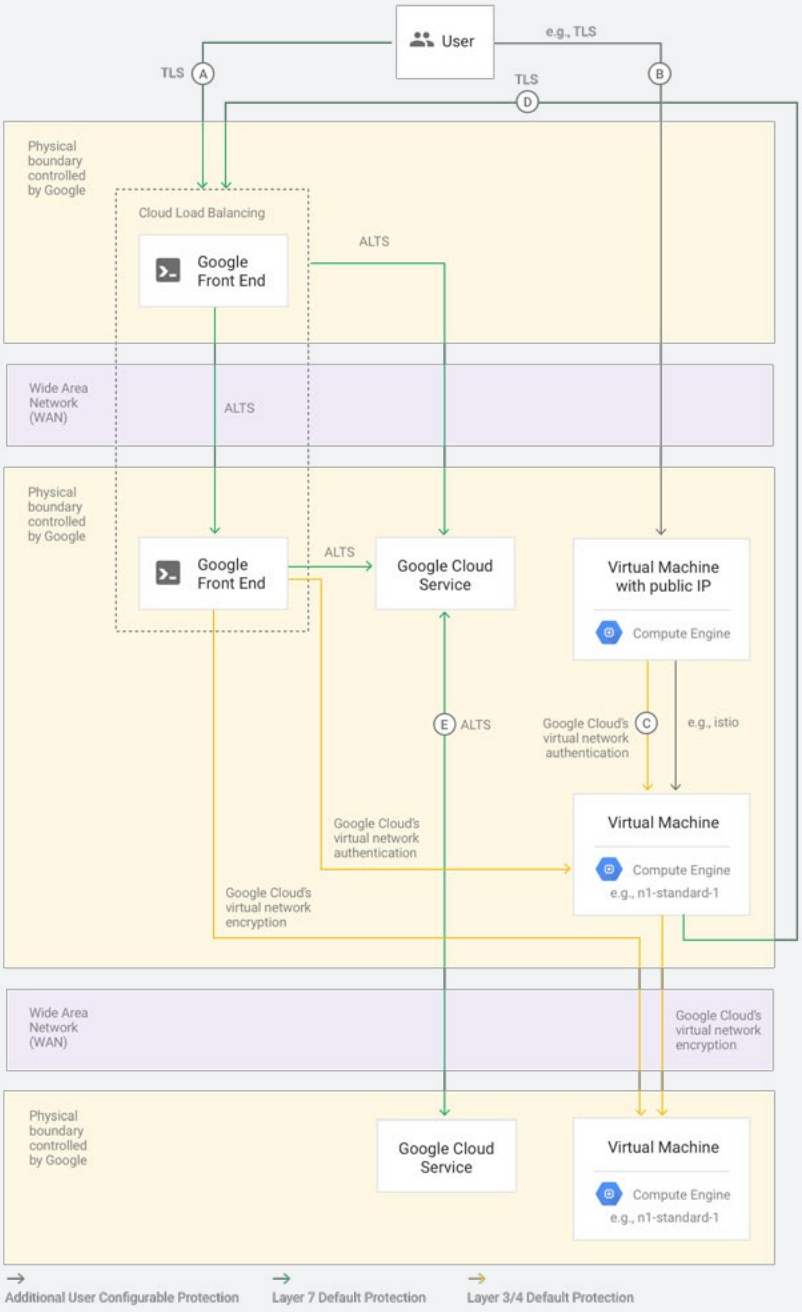
GFEs proxy traffic to Google Cloud services. GFEs route the user's request over our network backbone to a Google Cloud service. This connection is authenticated and encrypted from GFE to the front-end of the Google Cloud service or customer application, when those communications leave a physical boundary controlled by Google or on behalf of Google. Figure 1 shows this interaction (labelled connection A).

End user (Internet) to a customer application hosted on Google Cloud

There are several ways traffic from the Internet can be routed to a customer application you host on Google Cloud. The way your traffic is routed depends on your configuration, as explained below. Figure 1 shows this interaction (labelled connection B).

- **Using a Google Cloud HTTP(S) or TCP/SSL proxy Load Balancer external load balancer:** A customer application hosted on Google Compute Engine VMs can use a Google Cloud Load Balancer

[1] Partner solutions include both solutions offered in Cloud Launcher, as well as products built in collaboration with partners, such as Cloud Dataprep.



[Figure 1]

Protection by default and options overlaid on Google's network

(GCLB) service to terminate HTTP(S), TLS, or TCP connections and to proxy, route, and distribute this traffic to their VMs. These load balancer services are implemented by the GFEs, much as GFEs terminate and route traffic for Google Cloud services. When GCLB routes traffic between GFEs, the connections are

authenticated, and encrypted when the traffic leaves a physical boundary controlled by or on behalf of Google. When GCLB routes traffic between a GFE and a physical machine that hosts a customer's VM, this traffic is authenticated and encrypted, when it leaves a physical boundary controlled by or on behalf of Google.

- **Using a Google Cloud HTTP(S) or TCP/SSL proxy Load Balancer external load balancer**: A customer application hosted on Google Compute Engine VMs can use a Google Cloud Load Balancer (GCLB) service to terminate HTTP(S), TLS, or TCP connections and to proxy, route, and distribute this traffic to their VMs. These load balancer services are implemented by the GFEs, much as GFEs terminate and route traffic for Google Cloud services. When GCLB routes traffic between GFEs, the connections are authenticated, and encrypted when the traffic leaves a physical boundary controlled by or on behalf of Google. When GCLB routes traffic between a GFE and a physical machine that hosts a customer's VM, this traffic is authenticated and encrypted, when it leaves a physical boundary controlled by or on behalf of Google.

  For HTTPS load balancers, connections between end users and the GFE are encrypted and authenticated with TLS or QUIC, using certificates that customers provide for the load balancer. For HTTP load balancers, connections between end users and GFE are not encrypted or authenticated.

  For SSL load balancers, connections between end users and the GFE are encrypted with TLS, similarly using customer-provided certificates. For TCP load balancers, there is no encryption between the end user and the GFE. The customer's application may, however, use its own encryption between the end user and the VMs.

- **Using a connection directly to a VM using an external IP or network load balancer IP:** If you are connecting via the the VM's external IP, or via a network-load-balanced IP, the connection does not go through the GFE. This connection is not encrypted by default and its security is provided at the user's discretion.

- **Using Cloud VPN:** If you are connecting from a host on your premises to a Google Cloud VM via a VPN, the connection goes from/to your on-premises host, to the on-premises VPN, to the

Google VPN, to the Google Cloud VM; the connection does not go through the GFE. The connection is protected from the on-premises VPN to the Google VPN with IPsec. The connection from the Google VPN to the Google Cloud VM is authenticated and encrypted, when those communications leave a physical boundary controlled by or on behalf of Google.

- **Using Cloud Dedicated Interconnect:** If you are connecting via Dedicated Interconnect, the connection goes from/to your on-premises host directly and the connection does not go through the GFE. This connection is not encrypted by default and its security is provided at the user's discretion.

Virtual Machine to Virtual Machine

VM to VM routing that takes place on our network backbone, using RFC1918 private IP addresses, may require routing traffic outside of the physical boundaries controlled by or on behalf of Google. Examples of VM to VM routing include:

- Compute Engine VMs sending requests to each other
- A customer VM connecting to a Google-managed VM like Cloud SQL

VM to VM connections are encrypted if they leave a physical boundary, and are authenticated within the physical boundary. VM to VM traffic, using public IP addresses, is not encrypted by default and its security is provided at the user's discretion. Figure 1 shows this interaction (labelled connection C).

Virtual Machine to Google Cloud service

If a VM routes a request to a Google Cloud service, the request is routed to a GFE (except in cases where the Google Cloud service is running on a Google-managed VM, as discussed above). The GFE receives the request, then routes the request in the same way it does for requests coming from the Internet: for traffic from a VM to a Google Cloud service, this is routed through private Google paths to the same public IPs for the GFEs. Private Google access allows VMs without public IPs to access some Google Cloud services and customer applications hosted on Google App Engine. (Note that if a VM is connecting to a customer application hosted on Google Compute

Engine or Google Container Engine, that traffic is routed the same way requests coming from the Internet are routed, over external paths.) Figure 1 shows this interaction (labelled connection D). An example of this kind of routing request is between a Compute Engine VM to Google Cloud Storage, or to a Machine Learning API. Google Cloud services support protecting these connections with TLS by default[2]. This protection is in place from the VM to the GFE. The connection is authenticated from the GFE to the service and encrypted if the connection leaves a physical boundary.

Google Cloud service to Google Cloud service

Routing from one production service to another takes place on our network backbone and may require routing traffic outside of physical boundaries controlled by or on behalf of Google. Figure 1 shows this interaction (labelled connection E). An example of this kind of traffic is a Google Cloud Storage event triggering Google Cloud Functions. Connections between production services are encrypted if they leave a physical boundary, and authenticated within the physical boundary.

3. Encryption in Transit by Default

Google uses various methods of encryption, both default and user configurable, for data in transit. The type of encryption used depends on the OSI layer, the type of service, and the physical component of the infrastructure. Figures 2 and 3 below illustrate the optional and default protections Google Cloud has in place for layers 3, 4, and 7.

The remainder of this section describes the default protections that Google uses to protect data in transit.

3.1 User to Google Front End encryption

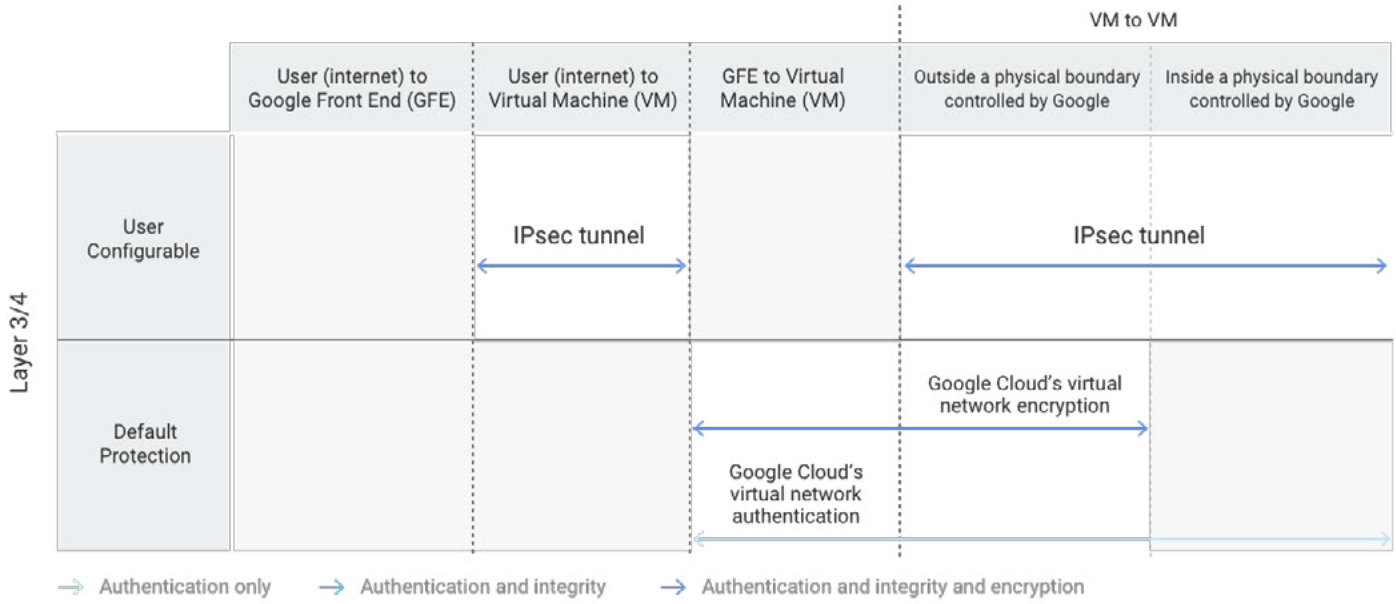
Today, many systems use the HTTPS protocol to communicate over the Internet. HTTPS provides security by directing the protocol over a TLS connection, ensuring the authenticity, integrity, and privacy of requests and responses. To accept HTTPS requests, the receiver requires a public–private key pair and an X.509 certificate, for server authentication, from a Certificate Authority (CA). The key pair and certificate help protect a user's requests at the application layer (layer 7) by proving that the receiver owns the domain name for which requests are intended. The following subsections discuss the components of user to GFE encryption, namely: TLS, BoringSSL, and

[2] You can still disable this encryption, for example for HTTP access to Google Cloud Storage buckets.

Google's Certificate Authority. Recall that not all customer paths route via the GFE; notably, the GFE is used for traffic from a user to a Google Cloud service, and from a user to a customer application hosted on Google Cloud that uses Google Cloud Load Balancing.

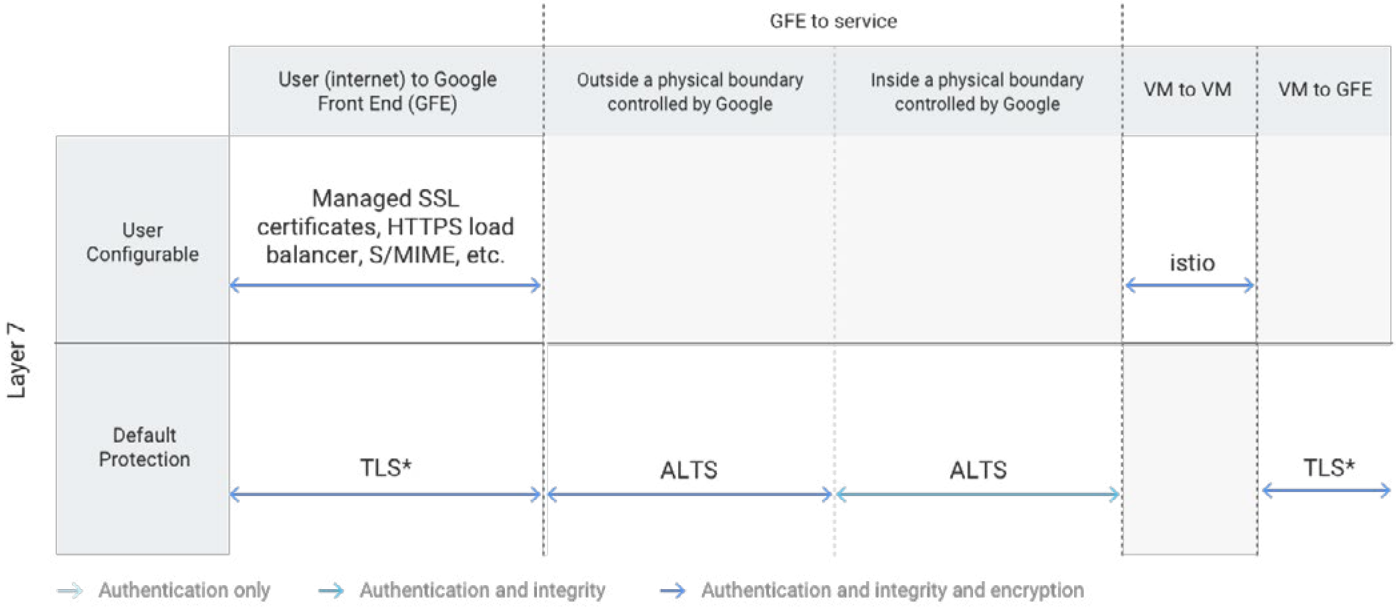
[Figure 2]

Protection by Default



[Figure3]

Protection by Default and Options at Layer 7 across Google Cloud[3]



* TLS is by default for Google Cloud services. For a customer application hosted on Google Cloud, this is something that needs to be configured by the customer.

[3] VM-to-Service communications not protected at Layer 7 are still protected at layers 3 and 4

3.1.1 Transport Layer Security (TLS)

When a user sends a request to a Google Cloud service, we secure the data in transit; providing authentication, integrity, and encryption, using the HTTPS protocol with a certificate from a web (public) certificate authority. Any data the user sends to the GFE is encrypted in transit with Transport Layer Security (TLS) or QUIC. GFE negotiates a particular encryption protocol with the client depending on what the client is able to support. GFE negotiates more modern encryption protocols when possible.

GFE's scaled TLS encryption applies not only to end-user interactions with Google, it also facilitates API interactions with Google over TLS, including Google Cloud. Additionally, our TLS encryption is used in Gmail to exchange email with external mail servers (more detail in section 4.2.2).

Google is an industry leader in both the adoption of TLS and the strengthening of its implementation. To this end, we have enabled, by default, many of the security features of TLS. For example, since 2011 we have been using forward secrecy in our TLS implementation. Forward secrecy makes sure the key that protects a connection is not persisted, so an attacker that intercepts and reads one message cannot read previous messages.

When a user sends a request to a Google Cloud service, we secure the data in transit; providing authentication, integrity, and encryption, using the HTTPS protocol with a certificate from a web (public) certificate authority.

3.1.2 BoringSSL

BoringSSL is a Google-maintained, open-source implementation of the TLS protocol, forked from OpenSSL, that is mostly interface-compatible with OpenSSL. Google forked BoringSSL from OpenSSL to simplify OpenSSL, both for internal use and to better support the Chromium and Android Open Source Projects. BoringCrypto, the core of BoringSSL, has been validated to FIPS 140-2 level 1.

TLS in the GFE is implemented with BoringSSL. Table 1 shows the encryption protocols that GFE supports when communicating with clients.

[Table 1]

Encryption Implemented in the Google Front End for Google Cloud Services and Implemented in the BoringSSL Cryptographic Library

Protocols	Authenticaion	Key Exchange	Encryption	Hash Functions
TLS 1.3[4] TLS 1.2 TLS 1.1 TLS 1.0[5] QUIC[6]	RSA 2048 ECDSA P-256	Curve25519 P-256 (NIST secp256r1)	AES-128-GCM AES-256-GCM AES-128-CBC AES-256-CBC ChaCha20-Poly1305 3DES[7]	SHA384 SHA256 SHA1[8] MD5[9]

3.1.3 Google's Certificate Authority

As part of TLS, a server must prove its identity to the user when it receives a connection request. This identity verification is achieved in the TLS protocol by having the server present a certificate containing its claimed identity. The certificate contains both the server's DNS hostname and its public key. Once presented, the certificate is signed by an issuing Certificate Authority (CA) that is trusted by the user requesting the connection[10]. As a result, users who request connections to the server only need to trust the root CA. If the server wants to be accessed ubiquitously, the root CA needs to be known to the client devices worldwide. Today, most browsers, and other TLS client implementations, each have their own set of root CAs that are configured as trusted in their "root store".

[4] TLS 1.3 is not yet finalized. The draft version is implemented only for certain Google domains for testing, such as Gmail.
[5] Google supports TLS 1.0 for browsers that still use this version of the protocol. Note that any Google site processing credit card information will no longer support TLS 1.0 by July 2018 when Payment Card Industry (PCI) compliance requires its deprecation.
[6] For details on QUIC, see https://www.chromium.org/quic.
[7-9] For backwards compatibility with some legacy operating systems, we support 3DES, SHA1 and MD5.
[10] In the case of chained certificates, the CA is transitively trusted.

Historically, Google operated its own issuing CA, which we used to sign certificates for Google domains. We did not, however, operate our own root CA. Today, our CA certificates are cross-signed by multiple root CAs which are ubiquitously distributed, including Symantec ("GeoTrust") and roots previously operated by GlobalSign ("GS Root R2" and "GS Root R4").

In June 2017, we announced a transition to using Google-owned root CAs. Over time, we plan to operate a ubiquitously distributed root CA which will issue certificates for Google domains and for our customers.

3.1.3.1 Root key migration and key rotation

Root CA keys are not changed often, as migrating to a new root CA requires all browsers and devices to embed trust of that certificate, which takes a long time. As a result, even though Google now operates its own root CAs, we will continue to rely on multiple third-party root CAs for a transitional period to account for legacy devices while we migrate to our own.

Creating a new root CA requires a key ceremony. At Google, the ceremony mandates that a minimum 3 of the 6 possible authorized individuals physically gather to use hardware keys that are stored in a safe. These individuals meet in a dedicated room, shielded from electromagnetic interference, with an air-gapped Hardware Security Module (HSM), to generate a set of keys and certificates. The dedicated room is in a secure location in Google data centers. Additional controls, such as physical security measures, cameras, and other human observers, ensure that the process goes as planned. If the ceremony is successful the generated certificate is identical to a sample certificate, except for the issuer name, public key and signature. The resulting root CA certificate is then submitted to browser and device root programs for inclusion. This process is designed to ensure that the privacy and security of the associated private keys are well understood so the keys can be relied upon for a decade or more.

As described earlier, CAs use their private keys to sign certificates, and these certificates verify identities when initiating a TLS handshake as part of a user session. Server certificates are signed with intermediate CAs, the creation of which is similar to the creation of a root CA. The intermediate CA's certificates are distributed as part of

the TLS session so it's easier to migrate to a new intermediate CA. This method of distribution also enables the CA operator to keep the root CA key material in a offline state.

The security of a TLS session is dependent on how well the server's key is protected. To further mitigate the risk of key compromise, Google's TLS certificate lifetimes are limited to approximately three months and the certificates are rotated approximately every two weeks.

A client that has previously connected to a server can use a private ticket key[11] to resume a prior session with an abbreviated TLS handshake, making these tickets very valuable to an attacker. Google rotates ticket keys at least once a day and expires the keys across all properties every 3 days. To learn more about session key ticket rotation, see Measuring the Security Harm of TLS Crypto Shortcuts.

3.2 Google Front End to Application Front Ends

In some cases, as discussed in section 2.2, the user connects to a GFE inside of a different physical boundary than the desired service and the associated Application Front End. When this occurs, the user's request and any other layer 7 protocol, such as HTTP, is either protected by TLS, or encapsulated in an RPC which is protected using Application Layer Transport Security (ALTS), discussed in section 3.4. These RPCs are authenticated and encrypted.

3.3 Google Cloud's virtual network encryption and authentication

Google Cloud's virtual network infrastructure enables encryption when traffic goes outside our physical boundaries. Encryption is performed at the network layer and applies to private IP traffic within the same Virtual Private Cloud (VPC) or across peered VPC networks.

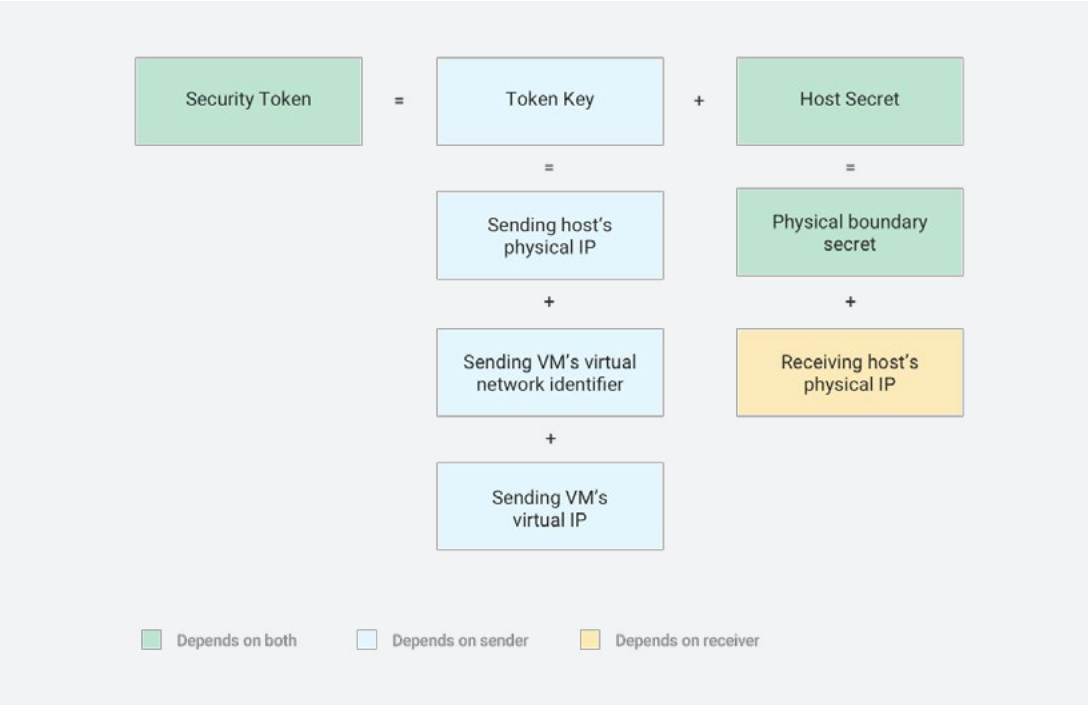
We assume that any network crossing a physical boundary not controlled by or on behalf of Google can be compromised by an active adversary, who can snoop, inject, or alter traffic on the wire. We ensure the integrity and privacy of communications using encryption when data moves outside physical boundaries we don't control.

Google Cloud's virtual network infrastructure enables encryption when traffic goes outside our physical boundaries. Encryption is performed at the network layer.

[11] This could be either a session ticket (RFC 5077) or a session ID (RFC 5246).

For Google Cloud services, RPCs are protected using ALTS by default. For customer applications hosted on Google Cloud, if traffic is routed via the Google Front End, for example if they are using the Google Cloud Load Balancer, traffic to the VM is protected using Google Cloud's virtual network encryption, described in the next section.

We use the Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) with a 128 bit key (AES-128-GCM) to implement encryption at the network layer. Each pair of communicating hosts establishes a session key via a control channel protected by ALTS for authenticated and encrypted communications. The session key is used to encrypt all VM-to-VM communication between those hosts, and session keys are rotated periodically.



[Figure 4]

Security Tokens

At the network layer (layer 3), Google Cloud's virtual network authenticates all traffic between VMs. This authentication, achieved via security tokens, protects a compromised host from spoofing packets on the network.

During authentication, security tokens are encapsulated in a tunnel header which contains authentication information about the sender and receiver. The control plane[12] on the sending side sets the token, and the receiving host validates the token. Security tokens are pre-generated for every flow, and consist of a token key (containing the sender's information) and the host secret. One secret exists for every source-receiver pair of physical boundaries controlled by or on behalf of Google. Figure 4 shows how token keys, host secrets, and security tokens are created.

The physical boundary secret is a 128-bit pseudorandom number, from which host secrets are derived by taking an HMAC-SHA1. The physical boundary secret is negotiated by a handshake between the network control planes of a pair of physical boundaries and renegotiated every few hours. The security tokens used for individual VM-to-VM authentication, derived from these and other inputs, are HMACs, negotiated for a given sender and receiver pair.

3.4 Service-to-service authentication, integrity, and encryption

1Within Google's infrastructure, at the application layer (layer 7), we use our Application Layer Transport Security (ALTS) for the authentication, integrity, and encryption of Google RPC calls from the GFE to a service, and from service to service.

ALTS uses service accounts for authentication. Each service that runs in Google's infrastructure runs as a service account identity with associated cryptographic credentials. When making or receiving RPCs from other services, a service uses its credentials to authenticate. ALTS verifies these credentials using an internal certificate authority.

Within a physical boundary controlled by or on behalf of Google, ALTS provides both authentication and integrity for RPCs in "authentication and integrity" mode. For traffic over the WAN outside of physical boundaries controlled by or on behalf of Google, ALTS enforces encryption for infrastructure RPC traffic automatically in "authentication, integrity, and privacy" mode. Currently, all traffic to Google services, including Google Cloud services, benefits from these same protections.

Within Google's infrastructure, at the application layer, we use our Application Layer Transport Security (ALTS) for the authentication, integrity, and encryption of Google RPC calls from the GFE to a service, and from service to service.

[12] **The control plane** is the part of the network that carries signalling traffic and is responsible for routing.

ALTS is also used to encapsulate other layer 7 protocols, such as HTTP, in infrastructure RPC mechanisms for traffic moving from the Google Front End to the Application Front End. This protection isolates the application layer and removes any dependency on the network path's security.

Services can be configured to accept and send ALTS communications only in "authentication, integrity and privacy" mode, even within physical boundaries controlled by or on behalf of Google. One example is Google's internal key management service, which stores and manages the encryption keys used to protect data stored at rest in Google's infrastructure.

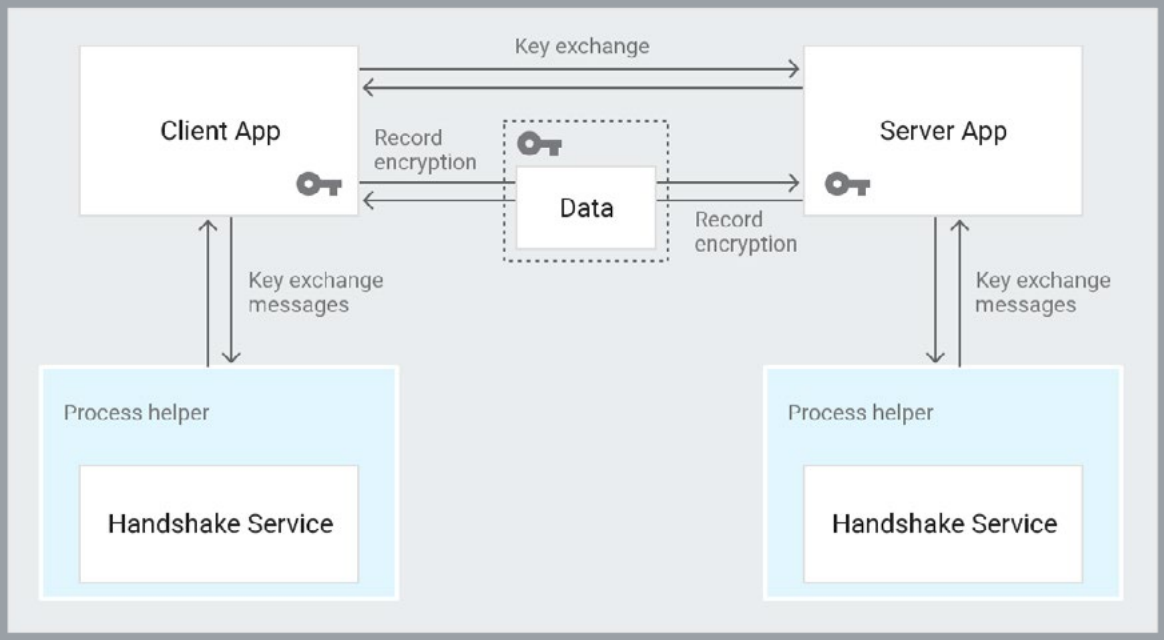
3.4.1 ALTS Protocol

ALTS has a secure handshake protocol similar to mutual TLS. Two services wishing to communicate using ALTS employ this handshake protocol to authenticate and negotiate communication parameters before sending any sensitive information. The protocol is a two-step process:

Step 1: Handshake
The client initiates an elliptic curve-Diffie Hellman (ECDH) handshake with the server using Curve25519. The client and server each have certified ECDH public parameters as part of their certificate, which is used during a Diffie Hellman key exchange. The handshake results in a common traffic key that is available on the client and the server. The peer identities from the certificates are surfaced to the application layer to use in authorization decisions.

Step 2: Record encryption
Using the common traffic key from Step 1, data is transmitted from the client to the server securely. Encryption in ALTS is implemented using BoringSSL and other encryption libraries. Encryption is most commonly AES-128-GCM while integrity is provided by AES-GCM's GMAC.

Figure 5 below shows the ALTS handshake in detail. In newer implementations, a process helper does the handshake; there are still some cases where this is done directly by the applications.



[Figure 5]

ALTS Handshake

As described at the start of section 3.4, ALTS uses service accounts for authentication, with each service that runs on Google's infrastructure running as a service identity with associated cryptographic credentials. During the ALTS handshake, the process helper accesses the private keys and corresponding certificates that each client-server pair uses in their communications. The private key and corresponding certificate (signed protocol buffer) have been provisioned for the service account identity of the service.

ALTS Certificates

There are multiple kinds of ALTS certificate:

- **Machine certificates:** provide an identity to core services on a specific machine. These are rotated approximately every 6 hours.

- **User certificates:** provide an end user identity for a Google engineer developing code. These are rotated approximately every 20 hours.
- **Borg job certificates:** provide an identity to jobs running within Google's infrastructure. These are rotated approximately every 48 hours.

The root certification signing key is stored in Google's internal certificate authority (CA), which is unrelated and independent of our external CA.

3.4.2 Encryption in ALTS

Encryption in ALTS can be implemented using a variety of algorithms, depending on the machines that are used. For example, most services use AES-128-GCM[13]. More information on ALTS encryption can be found in Table 2.

[Table 2]

Encryption in ALTS

Machines	Message encryption used	
Most common	AES-128-GCM	
Sandy Bridge or older	AES-128-VCM	Uses a VMAC instead of a GMAC and is slightly more efficient on these older machines.

Most Google services use ALTS, or RPC encapsulation that uses ALTS. In cases where ALTS is not used, other protections are employed. For example:

- Some low-level machine management and bootstrapping services use SSH
- Some low-level infrastructure logging services TLS or Datagram TLS (DTLS)[14]
- Some services that use non-TCP transports use other cryptographic protocols or network level protections when inside physical boundaries controlled by or on behalf of Google

[13] Previously, other protocols were used but are now deprecated. Less than 1% of jobs use these older protocols.
[14] Datagram TLS (DTLS) provides security for datagram-based applications by allowing them to communicate in a way that prevents eavesdropping and tampering.

Communications between VMs and Google Cloud Platform services use TLS to communicate with the Google Front End, not ALTS. We describe these communications in section 3.5.

3.5 Virtual machine to Google Front End encryption

VM to GFE traffic uses external IPs to reach Google services, but you can configure the Private Google Access feature to use Google-only IP addresses for the requests.

As with requests from an external user to Google, we support TLS traffic by default from a VM to the GFE. The connection happens in the same way as any other external connection. For more information on TLS, see section 3.1.1.

4. User-configurable options for encryption in transit

Section 3 of this document described the default protections that Google has in place for data in transit. This section describes the configurations our users can make to these default protections.

4.1 On-premises data center to Google Cloud

4.1.1 TLS using GCLB external load balancers

If your cloud service uses a Google HTTPS or SSL Proxy external load balancer, then GFE terminates the TLS connections from your users using SSL certificates that you provision and control. More information on customizing your certificate can be found in our SSL Certificates documentation.

4.1.2 IPsec tunnel using Google Cloud VPN

As a Google Cloud customer, you can use Google Cloud VPN to securely connect your on-premises network to your Google Cloud Platform Virtual Private Cloud (VPC) network through an IPsec VPN

connection (layer 3). Traffic traveling between the two networks is encrypted by one VPN gateway and decrypted by the other VPN gateway. This protects your data over the Internet. In addition, you can set up multiple, load-balanced tunnels through multiple VPN gateways. The Google Cloud VPN protects your data in the following ways:

- Packets **from your VMs to the Cloud VPN** remain within Google's network. These packets are encrypted by Google Cloud's virtual network if they travel outside the physical boundaries controlled by or on behalf of Google.
- Packets **from the Cloud VPN to your on-premises VPN** are encrypted and authenticated using an IPsec tunnel.
- Packets **from your on-premises VPN to your on-premises hosts** are protected by whatever controls you have in place on your network.

To set up a VPN, create a Cloud VPN gateway and tunnel on the hosted service's VPC network, then permit traffic between the networks. You also have the option of setting up a VPN between two VPCs.

You can further customize your network by specifying the Internet Key Exchange (IKE)[15] version for your VPN tunnel. There are two versions of IKE to choose from, IKEv1 and IKEv2, each of which supports different ciphers. If you specify IKEv1, Google encrypts the packets using AES-128-CBC and provides integrity through SHA-1 HMAC[16]. For IKEv2, a variety of ciphers are available and supported. In all cases, Google Cloud VPN will negotiate the most secure common protocol the peer devices support. Full instructions on setting up a VPN can be found in our documentation Creating a VPN.

An alternative to an IPsec tunnel is Google Cloud Dedicated Interconnect. Dedicated Interconnect provides direct physical connections and RFC1918 communication between your on-premises network and Google's network. The data traveling over this connection is NOT encrypted by default and so, should be secured at the application layer, using TLS for example. Google Cloud VPN and Google Cloud Interconnect use the same attachment point so you

As a Google Cloud customer, you can use Google Cloud VPN to securely connect your on-premises network to your Google Cloud Platform Virtual Private Cloud (VPC) network through an IPsec VPN connection.

[15] Internet Key Exchange (IKE) is the protocol used to set up a security association in the IPsec protocol suite.
[16] HMAC-SHA-1 is not broken by a SHA-1 collision, such as the SHAttered collision Google researchers found.

can use IPsec VPN encryption with Dedicated Interconnect however, to achieve this, you will need to use a third party solution. MACsec (layer 2 protection) is not currently supported.

4.2 User to Google Front End

4.2.1 Managed SSL certificates: Free and automated certificates

When building an application on Google Cloud, you can leverage GFE's support of TLS by configuring the SSL certificate you use. For example, you can have the TLS session terminate in your application. This termination is different to the TLS termination described in section 4.1.1.

When building an application on Google Cloud, you can leverage GFE's support of TLS by configuring the SSL certificate you use.

Google also provides free and automated SSL certificates in both the Firebase Hosting and Google App Engine custom domains. These certificates are only available for Google-hosted properties. With Google App Engine custom domains, you can also provide your own SSL certificates and use an HTTPS Strict Transport Protocol (HSTS) header.

Once your domain is pointed at Google's infrastructure, we request and obtain a certificate for that domain to allow secure communications. We manage the TLS server private keys, which are either 2048-bit RSA or secp256r1 ECC, and renew certificates on behalf of our customers.

4.2.2 Require TLS in Gmail

As discussed in section 3.1.1, Gmail uses TLS by default. Gmail records and displays whether the last hop an email made was over a TLS session[17]. When a Gmail user exchanges an email with another Gmail user, the emails are protected by TLS, or in some cases, sent directly within the application. In these cases, the RPCs used by the Gmail application are protected with ALTS as described in section 3.4. For incoming messages from other email providers, Gmail does not enforce TLS. Gmail administrators can configure Gmail to require a secure TLS connection for all incoming and outgoing emails.

[17] For G Suite enterprise, this isn't shown in the UI. Domain administrators can examine data for their domain using Email Log Search.

4.2.3 Gmail S/MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME) is an email security standard that provides authentication, integrity, and encryption. The implementation of the S/MIME standard mandates that certificates associated with users sending emails are hosted in a public CA.

As an administrator, you can configure Gmail to enable S/MIME for outgoing emails, set up policies for content and attachment compliance, and create routing rules for incoming and outgoing emails. Once configured, you must upload users' public certificates to Gmail using the Gmail API. For users external to Gmail, an initial S/MIME-signed message must be exchanged to set S/MIME as the default.

4.3 Service-to-service and VM-to-VM encryption

Istio is an open-source service mesh developed by Google, IBM, Lyft, and others, to simplify service discovery and connectivity. Istio authentication provides automatic encryption of data in transit between services, and management of associated keys and certificates. Istio can be used in Google Container Engine and Google Compute Engine.

If you want to implement mutual authentication and encryption for workloads, you can use istio auth. Specifically, for a workload in Kubernetes, Istio auth allows a cluster-level CA to generate and distribute certificates, which are then used for pod-to-pod mutual Transport Layer Security (mTLS).

Istio is an open-source service mesh designed to simplify service discovery and connectivity. Istio authentication provides automatic encryption of data in transit between services, and management of associated keys and certificates.

5. How Google helps the Internet encrypt data in transit

Sections three and four explained the default and customizable protections Google Cloud has in place for customer data in transit. In addition, Google has several open-source projects and other efforts that encourage the use of encryption in transit and data security on the Internet at large.

5.1 Certificate Transparency

As discussed in section 3.1, to offer HTTPS, a site must apply first for a certificate from a trusted web (public) Certificate Authority (CA). The Certificate Authority is responsible for verifying that the applicant is authorized by the domain holder, as well as ensuring that any other information included in the certificate is accurate.This certificate is then presented to the browser to authenticate the site the user is trying to access. In order to ensure HTTPS is properly authenticated, it's important to ensure that CAs only issue certificates that the domain holder has authorized.

Certificate Transparency (CT) is an effort that Google launched in March 2013 to provide a way for site operators and domain holders to detect if a CA has issued any unauthorized or incorrect certificates. It works by providing a mechanism for domain holders, CAs, and the public to log the trusted certificates they see or, in the case of CAs, the certificates they issue, to publicly verifiable, append-only, tamper-proof logs. The certificates in these logs can be examined by anyone to ensure the information is correct, accurate, and authorized.

The first version of Certificate Transparency was specified in an IETF experimental RFC, RFC 6962. During the development of Certificate Transparency, Google open-sourced a number of tools, including an open-source log server that can record certificates, as well as tools to create Certificate Transparency logs. In addition, Google Chrome requires that some certificates must be publicly disclosed, such as for Extended Validation (EV) certificates or certificates issued from CAs that have improperly issued certificates in the past. From 2018, Chrome will require that all new publicly trusted certificates be disclosed.

As a site operator, you can use Certificate Transparency to detect if unauthorized certificates have been issued for your website. A number of free tools exist to make this easy to do, such as Google's Certificate Transparency Report, Certificate Search, or tools from Facebook. Even if you don't use Certificate Transparency, a number of browsers now examine Certificate Transparency regularly to ensure that the CAs your users trust to access your website are adhering to industry requirements and best practices, reducing the risk of fraudulent certificates being issued.

5.2 Increasing the use of HTTPS

As described in section 3.1, we work hard to make sure that our sites and services provide modern HTTPS by default. Our goal is to achieve 100% encryption across our products and services. To this end, we publish an annual HTTPS Transparency Report that tracks our progress towards our goal for all properties, including Google Cloud. We continue to work through the technical barriers that make it difficult to support encryption in some of our products, such as solutions for browsers or other clients that do not support HTTPS Strict Transport Protocol (HSTS)[18]. We use HSTS for some of our sites, including the google.com homepage, to allow users to connect to a server only over HTTPS.

We know that the rest of the Internet is working on moving to HTTPS. We try to facilitate this move in the following ways:

- We provide developers with advice on why HTTPS matters, how to enable HTTPS, and best practices when implementing HTTPS
- We have created tools in Chrome like the Security panel in DevTools to help developers assess the HTTPs status of their site(s)
- We financially support the Let's Encrypt initiative that allows anyone to obtain a free certificate for their website. Google representatives sit on the technical advisory board of the Let's Encrypt's parent organization, Internet Security Research Group.

In 2016, we began publishing metrics on "HTTPS usage on the Internet" for the Top 100 non-Google sites on the Internet. With these metrics, we aim to increase awareness and help make the Internet a safer place for all users. In October 2017, Chrome formally renewed its financial support of Let's Encrypt as a Platinum sponsor.

Our goal is to achieve 100% encryption across our products and services. To this end, we publish an annual HTTPS Transparency Report that tracks our progress towards this our goal for all properties.

[18] HTTPS Strict Transport Protocol is a mechanism enabling web sites to declare themselves accessible only via secure connections and/or for users to be able to direct their user agent(s) to interact with given sites only over secure connections.

5.3 Increasing the use of secure SMTP: Gmail indicators

Most email is exchanged using the Simple Mail Transfer Protocol (SMTP) which, by default, sends email without using encryption. To encrypt an email, the mail provider must implement security controls like TLS.

As discussed in section 3.1, Gmail uses TLS by default. In addition, section 4.2.2 describes how Gmail administrators can enforce the use of TLS protection for incoming and outgoing emails. Like Google's efforts with HTTPS transparency, Gmail provides data on TLS use for incoming emails to Gmail. This data is presented in our Safer Email Transparency Report.

Google, in partnership with the IETF and other industry key players, is leading the development of SMTP STS. SMTP STS is like HSTS for HTTPS, forcing the use of SMTP over only encrypted channels.

5.4 Chrome APIs

In February 2015, Chrome announced that powerful new features will be available only to secure origins[19]. Such features include the handling of private information and access to sensors on a user's device. Starting with geolocation in Chrome 50, we began deprecating these features for insecure origins.

6. Ongoing Innovation in Encryption in Transit

6.1 Chrome Security User Experience

Google Chrome is an industry leader in leveraging its UI to display security information in ways that allow users to quickly understand the safety of their connection to a site. With this information, users can make informed decisions about when and how they share their data. Chrome conducts extensive user research, the results of which are shared in peer-reviewed papers.

Google Chrome is an industry leader in leveraging its UI to display security information in ways that allow users to quickly understand the safety of their connection to a site.

[19] Secure origins are connections that match certain scheme, host, or port patterns.

To help further protect its users, Chrome has announced that by the end of 2017, it will mark all HTTP connections as non-secure. Starting with Chrome 56, by default, users will see a warning if an HTTP page includes a form with password or credit card fields. With Chrome 62, a warning will be shown when a user enters in data on an HTTP page, and for all HTTP pages visited in Incognito mode. Eventually, Chrome will show a warning for all pages that are served over HTTP.

To see how particular configurations are displayed to users in Chrome, you can use the BadSSL tool.

6.2 Key Transparency

A significant deterrent to the widespread adoption of message encryption is the difficulty of public key exchange: how can I reliably find the public key for a new user with which I am communicating? To help solve this issue, in January 2017, Google announced Key Transparency. This is an open framework that provides a generic, secure, and auditable means to distribute public keys. The framework removes the need for users to perform manual key verification. Key Transparency is primarily targeted at the distribution of users' public keys in communications, for example, E2E and OpenPGP email encryption. Key Transparency's design is a new approach to key recovery and distribution and is based on insights gained from Certificate Transparency and CONIKS.

Key Transparency's development is open-source and it is implemented using a large-scale Merkle tree. Key Transparency Verification allows account owners to see what keys have been associated with their accounts and how long an account has been active and stable. The long-term goal of Google's Key Transparency work is to enable anyone to run a Key Transparency server and make it easy to integrate into any number of applications.

6.3 Post-quantum cryptography

Google plans to remain the industry leader in encryption in transit. To this end, we have started work in the area of post-quantum cryptography. This type of cryptography allows us to replace existing crypto primitives, that are vulnerable to efficient quantum attacks, with post-quantum candidates that are believed to be more robust. In

July 2016 we announced that we had conducted an experiment on the feasibility of deploying such an algorithm by using the New Hope post-quantum crypto algorithm in the developer version of Chrome. In addition to this work, researchers at Google have published papers on other practical post-quantum key-exchange protocols.

Appendix

For general information on Google Cloud security and compliance, see the security sections of the Google Cloud Platform website and the G Suite website, including the Google Infrastructure Security Design Overview and the public SOC3 audit report.

Application Layer Transport Security

A techinical whitepaper from Google Cloud

Cesar Ghali, Adam Stubblefield,
Ed Knapp, Jiangtao Li,
Benedikt Schmidt, Julien Boeuf

Google Cloud

Google Cloud

Table of Contents

Executive summary

- Google's Application Layer Transport Security (ALTS) is a mutual authentication and transport encryption system developed by Google and typically used for securing Remote Procedure Call (RPC) communications within Google's infrastructure. ALTS is similar in concept to mutually authenticated TLS but has been designed and optimized to meet the needs of Google's datacenter environments.

- The ALTS trust model has been tailored for cloud-like containerized applications. Identities are bound to entities instead of to a specific server name or host. This trust model facilitates seamless microservice replication, load balancing, and rescheduling across hosts.

- ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol. These protocols govern how sessions are established, authenticated, encrypted, and resumed.

- ALTS is a custom transport layer security solution that we use at Google. We have tailored ALTS to our production environment, so there are some tradeoffs between ALTS and the industry standard, TLS. Section 5 discusses these tradeoffs in more detail.

1. Introduction

Production systems at Google consist of a constellation of microservices[1] that collectively issue O(1010) Remote Procedure Calls (RPCs) per second. When a Google engineer schedules a production workload[2], any RPCs issued or received by that workload are protected with ALTS by default. This automatic, zero-configuration protection is provided by Google's Application Layer Transport Security (ALTS). In addition to the automatic protections conferred on RPC's, ALTS also facilitates easy service replication, load balancing, and rescheduling across production machines. This paper describes ALTS and explores its deployment over Google's production infrastructure.

Audience: This document is aimed at infrastructure security professionals who are curious about how authentication and transport security are performed at scale in Google.

Prerequisites: In addition to this introduction, we assume a basic understanding of cluster management at Google.

When a Google engineer schedules a production workload, any RPCs that workload issues has automatic, zero-configuration protection using Google's Application Layer Transport Security (ALTS).

2. Application-Level Security and ALTS

Many applications, from web browsers to VPNs, rely on secure communication protocols, such as TLS (Transport Layer Security) and IPSec, to protect data in transit[3]. At Google, we use ALTS, a mutual authentication and transport encryption system that runs at the application layer, to protect RPC communications. Using application-level security allows applications to have authenticated remote peer identity, which can be used to implement fine-grained authorization policies.

2.1 Why Not TLS?

It may seem unusual for Google to use a custom security solution such as ALTS when the majority of Internet traffic today is encrypted using TLS. ALTS began development at Google in 2007. At the time, TLS was bundled with support for many legacy protocols that did not satisfy our minimum security standards. We could have designed our security solution by adopting the TLS components we needed and implementing the ones we wanted; however, the advantages of

[1] A **microservice** is an architectural style that structures an application as a collection of loosely coupled services which implement business capabilities.
[2] A **production workload** is an application that Google engineers schedule to run in Google's datacenters.
[3] For more information on how Google protects data in transit, see our whitepaper, "Encryption in Transit in Google Cloud".

building a more Google-suited system from scratch outweighed the benefits of patching an existing system. In addition, ALTS is more appropriate for our needs, and historically more secure than older TLS. Listed below are the key differences between TLS and ALTS.

- There is a significant difference between the trust models[4] of TLS with HTTPS semantics and ALTS. In the former, server identities are bound to a specific name and corresponding naming scheme. In ALTS, the same identity can be used with multiple naming schemes. This level of indirection provides more flexibility and greatly simplifies the process of microservice replication, load balancing, and rescheduling between hosts.

- Compared to TLS, ALTS is simpler in its design and implementation. As a result, it is easier to monitor for bugs and security vulnerabilities using manual inspection of source code or extensive fuzzing.

- ALTS uses Protocol Buffer to serialize its certificates and protocol messages, while TLS uses X.509 certificates encoded with ASN.1. The majority of our production services use protocol buffers for communication (and sometimes storage), making ALTS a better fit for Google's environment.

2.2 ALTS Design

ALTS is designed to be a highly reliable, trusted system that allows for service-to-service authentication and security with minimal user involvement. To achieve this, the properties listed below are part of ALTS's design:

- **Transparency:** ALTS configuration is transparent to the application layer. By default, service RPCs are secured using ALTS. This allows application developers to focus on the functional logic of their services without having to worry about credential management or security configurations. During service-to-service connection establishment, ALTS provides applications with an authenticated remote peer identity which can be used for fine-grained authorization checks and auditing.

- **State-of-the-art cryptography:** All cryptographic primitives and protocols used by ALTS are up-to-date with current known attacks. ALTS runs on Google-controlled machines, meaning that

ALTS is designed to be a highly reliable, trusted system that allows for service-to-service authentication and security with minimal user involvement.

[4] A trust model is the mechanism through with a security protocol identifies, distributes and rotates credentials and identities.

all supported cryptographic protocols can be easily upgraded and quickly deployed.

- **Identity model:** ALTS performs authentication primarily by identity rather than host name. At Google, every network entity (e.g. a corporate user, a physical machine, or a production service or workload) has an associated identity. All communications between services are mutually authenticated.

- **Key distribution:** ALTS relies on each workload having an identity, which is expressed as a set of credentials. These credentials are deployed in each workload during initialization, without user involvement. In parallel, a root of trust and a trust chain for these credentials are established for machines and workloads. The system allows for automatic certificate rotation and revocation without application developers involvement.

- **Scalability:** ALTS is designed to be very scalable in order to support the massive scale of Google's infrastructure. This requirement resulted in the development of efficient session resumption, see Section 4.3.

- **Long-lived connections:** Authenticated key exchange cryptographic operations are computationally expensive. To accommodate the scale of Google's infrastructure, after an initial ALTS handshake, connections can be persisted for a longer time to improve overall system performance.

- **Simplicity:** TLS by default comes with support for legacy protocol versions and backwards compatibility. ALTS is considerably simpler as Google controls both clients and servers, which we designed to natively support ALTS.

3. ALTS Trust Model

ALTS performs authentication primarily by identity rather than host. At Google, every network entity (e.g., a corporate user, a physical machine, or a production service) has an associated identity. These identities are embedded in ALTS certificates and used for peer authentication during secure connection establishment. The model we pursue is that our production services run as production entities that can be managed by our Site Reliability Engineers (SREs)[5]. The

ALTS performs authentication primarily by identity rather than host. At Google, every network entity (e.g., a corporate user, a physical machine, or a production service) has an associated identity.

[5] Some services are managed directly by developers.

development versions of these production services run as test entities that can be managed by both SREs and developers.

For example, let's assume we have a product with two services: `service-frontend` and `service-backend`. SREs can launch the production version of these services: `service-frontend-prod` and `service-backend-prod`. Developers can build and launch development versions of these services, `service-frontend-dev` and `service-backend-dev`, for testing purposes. The authorization policy in the production services will be configured not to trust the development versions of the services.

3.1 ALTS Credentials

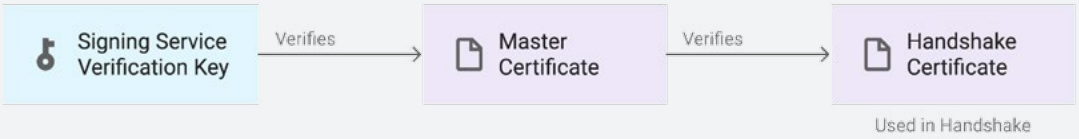
There are three types of ALTS credential, all of which are expressed in Protocol Buffer message format.

- **Master certificate:** signed by a remote Signing Service and used to verify handshake certificates. The master certificate contains a public key associated with a master private key, e.g., RSA keypair. This private key is used to sign handshake certificates. These certificates, when exercised in combination with the ALTS policy discussed below, are essentially constrained intermediate Certificate Authority (CA) certificates. Master certificates are typically issued for production machines and schedulers of containerized workloads such as the Borgmaster[6].

- **Handshake certificate:** created and signed locally by the master private key. This certificate contains the parameters used during the ALTS handshake (secure connection establishment), for example, static Diffie-Hellman (DH) parameters and the handshake ciphers. Also, the handshake certificate contains the master certificate that it is derived from, i.e., the one associated with the master private key that signs the handshake certificate.

- Resumption key: is a secret that is used to encrypt resumption tickets. This key is identified by a Resumption Identifier ID_R that is unique for, and shared among, all production workloads running with the same identity and in the same datacenter cell. For more details on session resumption in ALTS, see Section 4.3.

Figure 1 shows the ALTS certificate chain, which consists of a Signing Service verification key, a master certificate and a handshake certificate. The Signing Service verification keys are the root of trust in ALTS and are installed on all Google machines in our production and corporate networks.



[Figure 1]

ALTS certificate chain

In ALTS, a Signing Service certifies Master certificates which in turn certify Handshake certificates. As Handshake certificates are created more often than Master certificates, this architecture reduces the load on the Signing service. Certificate rotation happens frequently at Google, especially for handshake certificates[7]. This frequent rotation compensates for the static key exchange pairs carried by the handshake certificates[8].

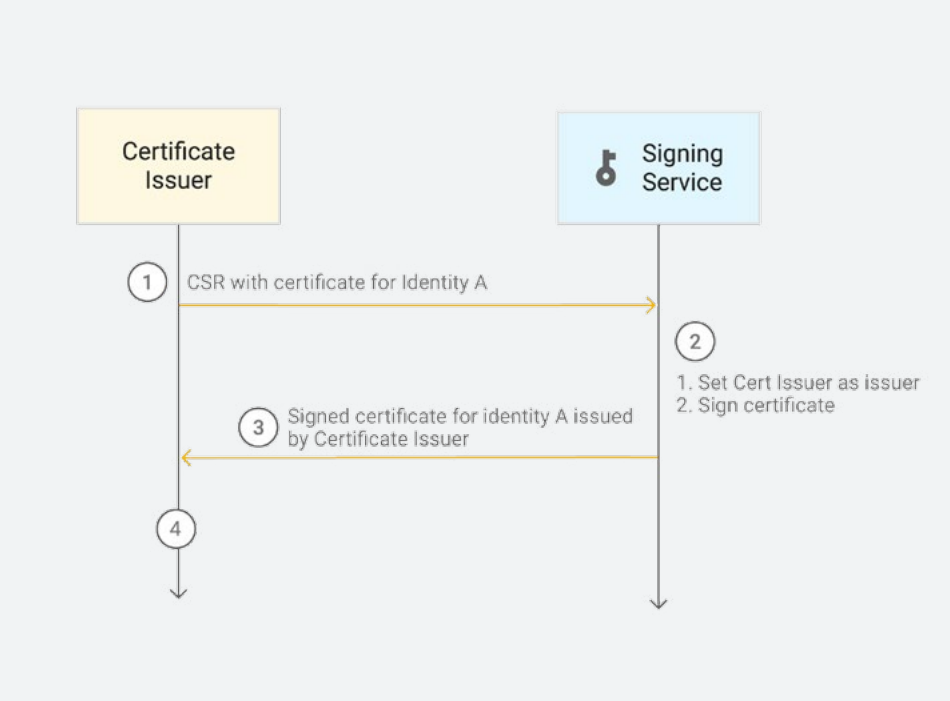
3.1.1 Certificate Issuance

In order to participate in an ALTS secure handshake, entities on the network need to be provisioned with handshake certificates. First, the issuer obtains a master certificate signed by the Signing Service and optionally passes it down to the entity. Then, a handshake certificate is created and signed by the associated master private key.

Typically, the issuer is our internal Certificate Authority (CA) when issuing certificates to machines and humans, or the Borgmaster when issuing certificates to workloads. However, it can be any other entity, e.g., a restricted Borgmaster for a test datacenter cell.

[6] Borgmaster is responsible for scheduling and initializing Google production workloads. For more information see Large-scale cluster management at Google with Borg.
[7] More information about certificate rotation frequencies can be found in "Encryption in Transit in Google Cloud".
[8] If a key is compromised, only the traffic for the lifetime of this keypair will be discoverable by the attacker.

Figure 2 shows how the Signing service is used to create a master certificate. The process consists of the following steps.



[Figure 2]

Certificate Issuance

1. The Certificate Issuer sends a Certificate Signing Request (CSR) to the Signing Service. This request asks the Signing Service to create a certificate for identity A. This identity, for example, can be a corporate user or the identity of a Google production service.

2. The Signing Service sets the issuer of the certificate (included in the CSR) to the requester (the Certificate Issuer in this case) and signs it. Recall that the corresponding Signing Service public (verifying) key is installed on all Google machines.

3. The Signing Service sends the signed certificate back.

4. A handshake certificate is created for identity A and is signed by the master certificate associated private key.

As shown in the process above, with ALTS, the issuer and signer of a certificate are two different logical entities. In this case, the issuer is the Certificate Issuer entity while the signer is the Signing Service.

There are three common categories of certificates in ALTS, namely: Human, Machine, and Workload. The following sections outline how each of these certificates are created and used in ALTS.

3.1.2 Human Certificates

At Google, we use ALTS to secure RPCs issued by human users to production services. To issue an RPC, a user must provide a valid handshake certificate. For example, if Alice wants to use an application to issue an ALTS-secure RPC, she can authenticate to our internal CA. Alice authenticates to the CA using her username, password, and two-factor authentication. This operation results in Alice getting a handshake certificate that is valid for 20 hours.

3.1.3 Machine Certificates

Every production machine in Google's datacenters has a machine master certificate. This certificate is used to create handshake certificates for core applications on that machine, e.g. machine management daemons. The primary identity embedded in a machine certificate refers to the typical purpose of the machine. For example, machines used to run different kinds of production and development workloads can have different identities. The master certificates are only usable by machines running verified software stacks; in some cases this trust is rooted in custom security hardware[9]. All production machine master certificates are issued by the CA and rotated every few months. Also, all handshake certificates are rotated every few hours.

3.1.4 Workload Certificates

A key advantage of ALTS is that it operates on the idea of a workload identity which facilitates easy service replication, load balancing, and rescheduling across machines. In our production network, we use a system called Borg[10] for cluster management and machine resource allocation at scale. The way that Borg issues certificates is part of the ALTS machine-independent workload identity implementation. The remainder of this section provides an overview of our workload certification.

Each workload in our production network runs in a Borg cell. Each cell contains a logically centralized controller called the Borgmaster, and several agent processes called Borglets that run on each

[9] Titan in depth: Security in plaintext.
[10] Large-scale cluster management at Google with Borg.

machine in that cell. Workloads are initialized with associated Workload Handshake Certificates issued by the Borgmaster. Figure 3 shows the process of workload certification in ALTS with Borg.

1. Each Borgmaster comes pre-installed with a Machine Master Certificate and associated private key (not shown in the diagram).

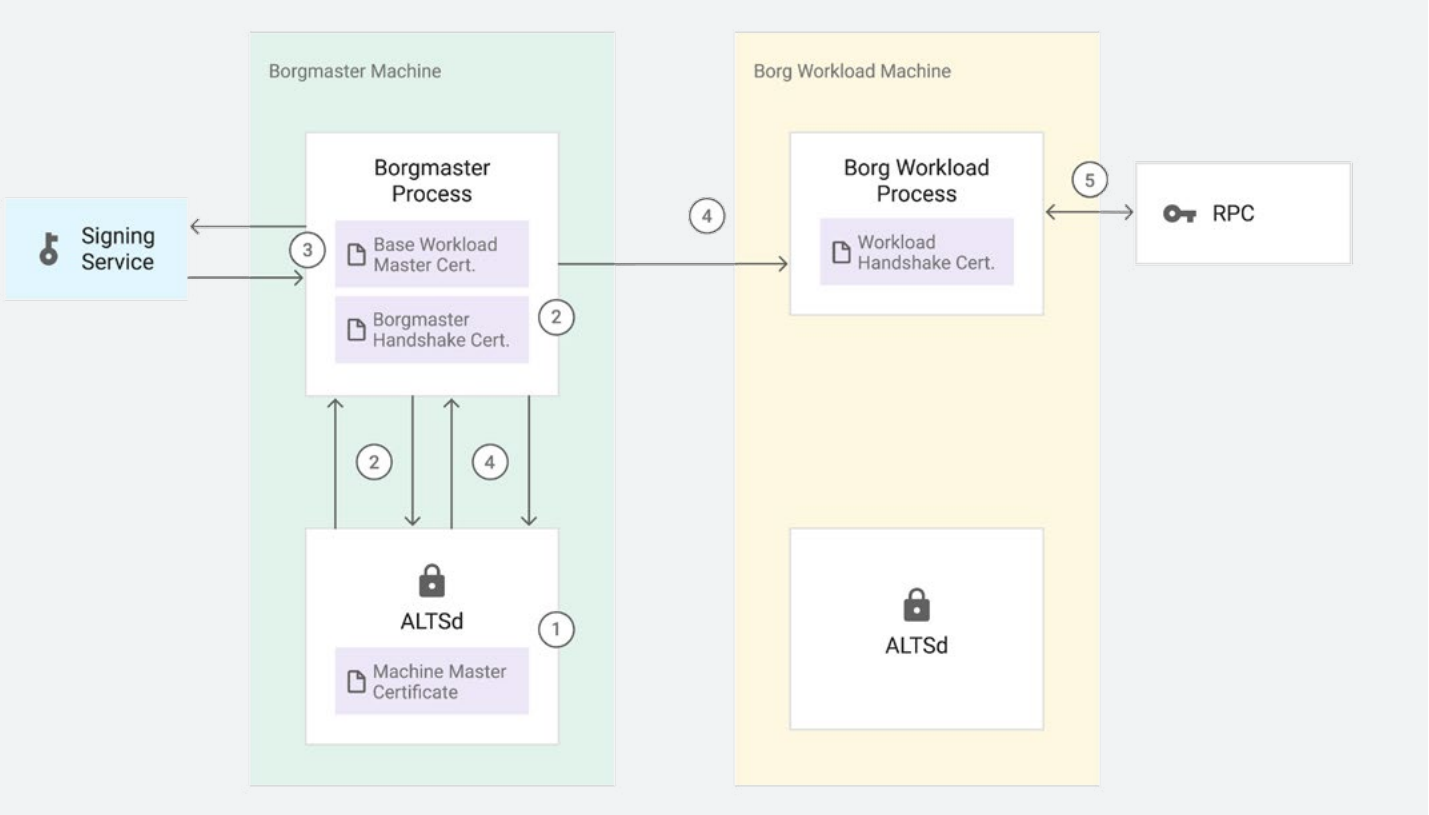
2. The ALTSd[11] generates a Borgmaster Handshake Certificate and signs it using the Machine Master private key. This Handshake Certificate allows Borgmaster to issue ALTS-secure RPCs.

3. The Borgmaster creates a Base Workload Master Certificate, and the corresponding private key. The Borgmaster initiates a request to get its Base Workload Master Certificate signed by the Signing Service. As a result, the Signing Service lists the Borgmaster as the issuer on this certificate.

The Borgmaster is now ready to schedule workloads that need to use ALTS. The steps below happen when a client schedules a workload to run on Borg as a given identity.

[Figure 3]

Handshake Certificate Creation in the Google Production Network



[11] ALTSd: a daemon responsible for, amongst other ALTS operations, the creation of handshake certificates.

4. The Borgmaster verifies that the client is authorized to run workloads as the identity that is specified in the workload configuration. If so, the Borgmaster schedules the Borg workload on the Borglet, and issues a Workload Handshake Certificate and its corresponding private key. This certificate is chained from the Base Workload Master Certificate. The Workload Handshake Certificate and its private key are then securely delivered to the Borglet (over a mutually authenticated ALTS protected channel between the Borgmaster and the Borglet). The Borgmaster rotates its Base Workload Master Certificate and reissues Handshake Certificates for all running workloads approximately every two days. In addition, each workload running as the same user in the same cell receives the same resumption key and identifier (ID_R) provisioned by the Borgmaster.

5. When the workload needs to make an ALTS-secure RPC, it uses the Workload Handshake Certificate in the handshake protocol. ID_R is also used as part of the handshake to initiate session resumption. For more information about session resumption in ALTS, see Section 4.3.

3.2 ALTS Policy Enforcement

The ALTS policy is a document that lists which issuers are authorized to issue certain categories of certificates for which identities. It is distributed to every machine on our production network. For example, the ALTS policy allows the CA to issue certificate to machines and humans. It also allows Borgmaster to issue certificates to workloads.

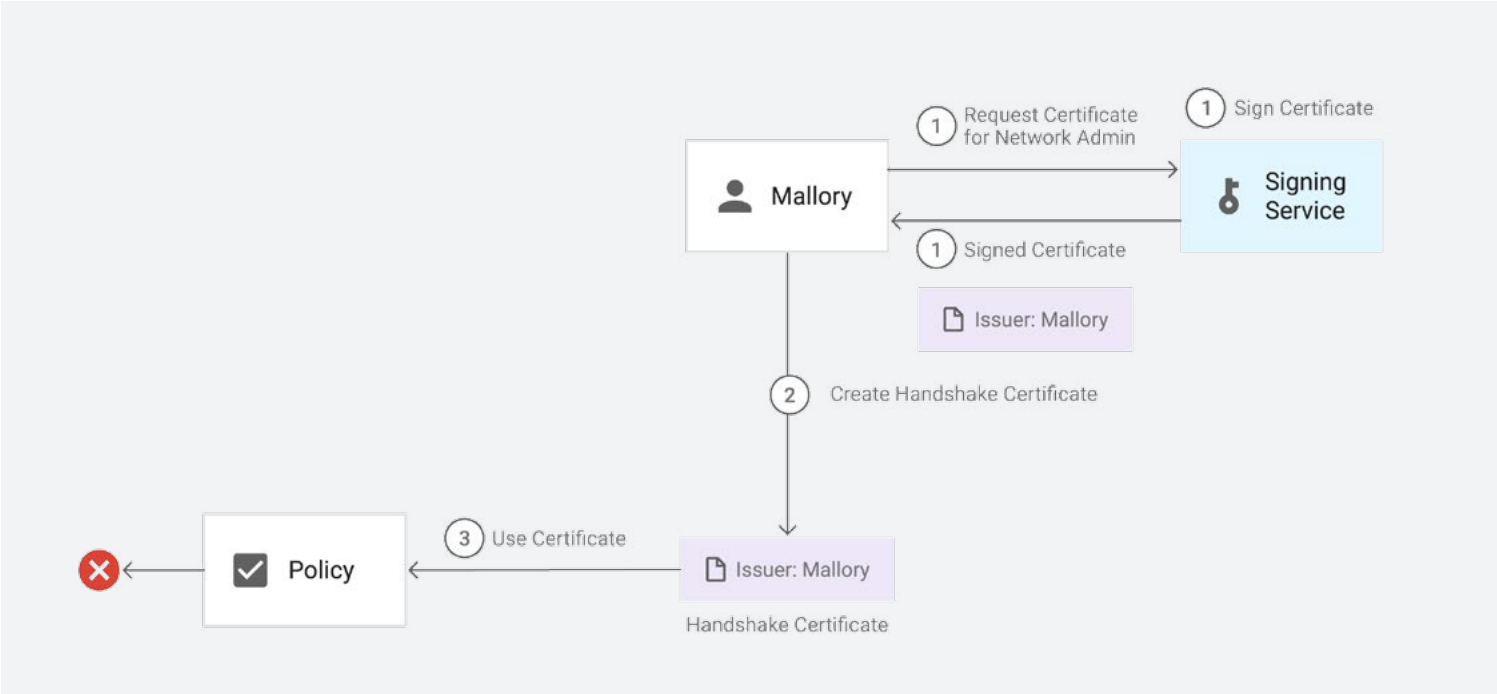
We have found that policy enforcement during certificate verification, as opposed to certificate issuance, is a more flexible approach as it allows for different policies to be enforced on different types of deployments. For example, we may want a policy in a test cluster to be more permissive than one in a production cluster.

During the ALTS handshake, the certificate validation includes a check of the ALTS policy. The policy ensures that the issuer listed in the certificate being validated is authorized to issue that certificate. If that is not the case, the certificate is rejected and the handshake

process fails. Figure 4 illustrates how the policy enforcement works in ALTS. Following the scenario in Figure 2, assume that Mallory (a corporate user who wants to escalate her privileges) wants to issue a master certificate to the Network Admin, which is a powerful identity that can reconfigure the network. It goes without saying that Mallory is not authorized on the ALTS policy to perform this operation.

[Figure 4]

Certificate Issuance and Usage



1. Mallory issues a master certificate for Network Admin identity and gets it signed by the Signing Service. This is similar to the first three steps in Figure 2.

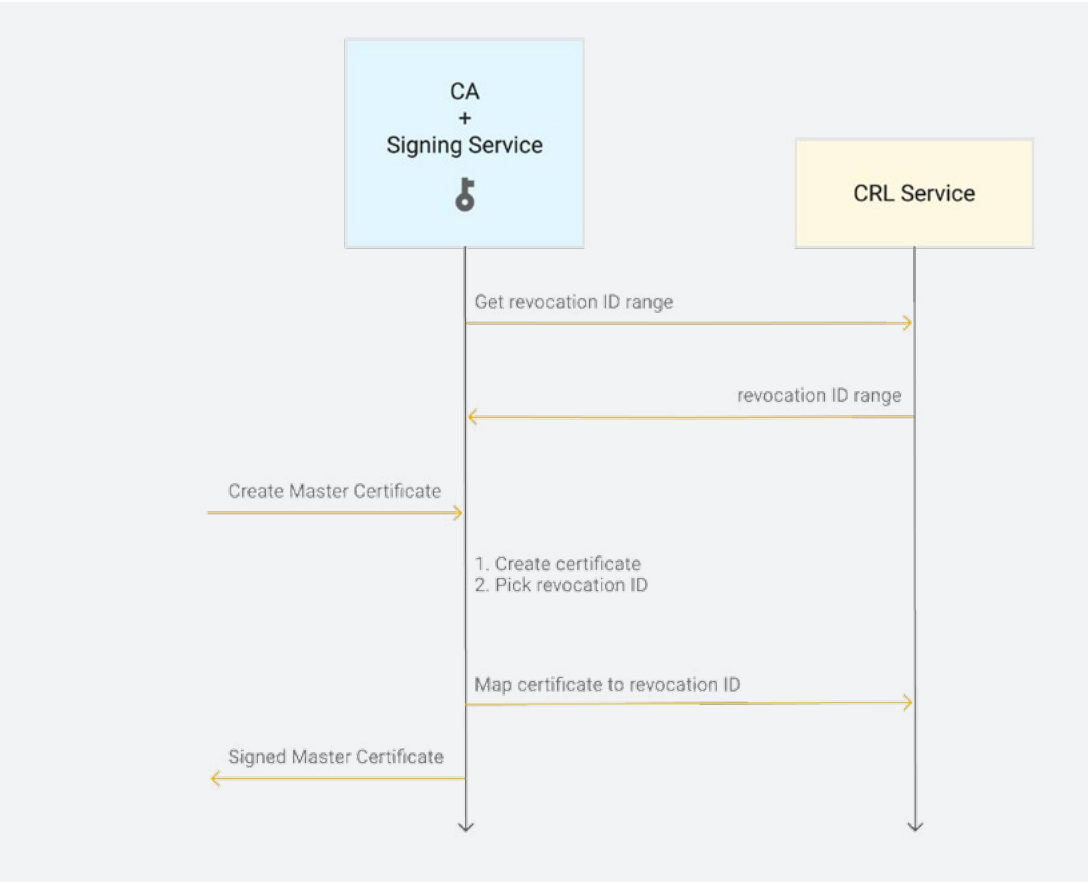
2. Mallory creates and signs a handshake certificate locally for Network Admin, using the master private key associated with the created master certificate.

3. If Mallory tries to impersonate the Network Admin identity by using the created handshake certificate, the ALTS policy enforcer, at the peer that Mallory tries to communicate with, will block the operation.

3.3 Certificate Revocation

At Google, a certificate is invalidated when it expires or it is included in our Certificate Revocation List (CRL). This section describes the design of Google's internal certificate revocation mechanisms, which, at the time of writing this paper, are still undergoing deployment testing.

All certificates issued to human corporate users have a daily expiration timestamp which forces the users to reauthenticate daily. Many of the certificates issued to production machines do not use expiration timestamps. We avoid relying on timestamps to expire production certificates as it can lead to outages caused by clock synchronization issues. Instead, we use the CRL as our source of truth for rotation and incident-response handling of certificates. Figure 5 shows how the CRL operates.



[Figure 5]

Master Certificate Creation with a Revocation ID

[12] In practice, the CA has access to the Signing Service private keys, making the two logical entities as a single physical one.

1. When an instance of our CA is initialized[12], it contacts the CRL Service and asks for a revocation ID range. A revocation ID is a 64-bit long ID with two components, an 8-bit certificate category (e.g. human or machine certificate), and a 56-bit certificate identifier. The CRL Service chooses a range of these IDs and returns it to the CA.

2. When the CA receives a request for a master certificate, it creates the certificate and embeds a revocation ID it picks from the range.

3. In parallel, the CA maps the new certificate to the revocation ID and sends this information to the CRL Service.

4. The CA issues the master certificate.

Revocation IDs assigned to handshake certificates depend on how the certificate is used. For example, handshake certificates that are issued to human corporate users inherit the revocation ID of the user's master certificate. For handshake certificates that are issued to Borg workloads, the revocation ID is assigned by the Borgmaster's range of revocation IDs. This ID range is assigned to the Borgmaster by the CRL Service in a process similar to that shown in Figure 5. Whenever a peer is involved in an ALTS handshake, it checks a local copy of the CRL file to ensure that the remote peer certificate has not been revoked.

The CRL Service compiles all revocation IDs into a single file that can be pushed to all Google machines that use ALTS. While the CRL database is several hundred megabytes, the generated CRL file is only a few megabytes due to a variety of compression techniques.

ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol.

4. ALTS Protocols

ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol. This section provides a high level overview of each protocol. These overviews should not be interpreted as detailed specifications of the protocols.

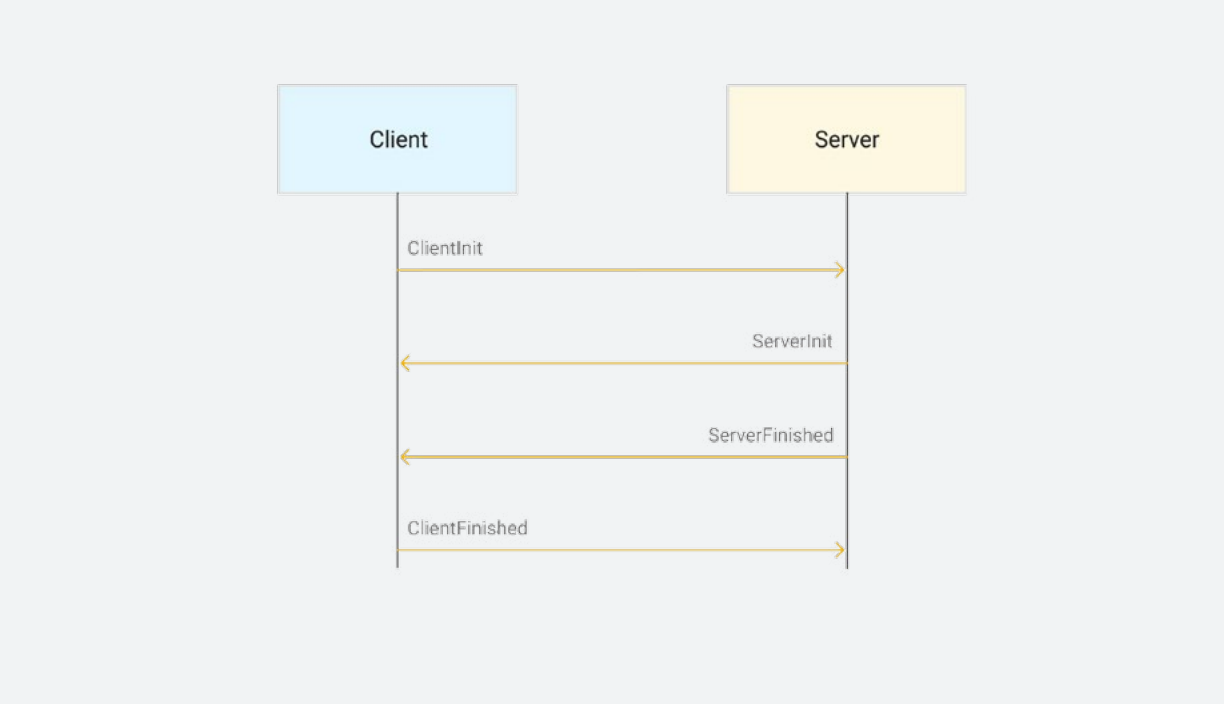
4.1 Handshake Protocol

The ALTS handshake protocol is a Diffie-Hellman-based authenticated key exchange protocol that supports both Perfect Forward Secrecy (PFS) and session resumption. The ALTS infrastructure ensures that each client and server have a certificate with their respective identities and an Elliptic Curve Diffie-Hellman (ECDH) key that chains to a trusted Signing Service verification key. In ALTS, PFS is not enabled by default because these static ECDH keys are frequently updated to renew forward secrecy even if PFS is not used on a handshake. During a handshake, the client and server securely negotiate a shared transit encryption key, and the Record protocol the encryption key will be used to protect. For example, the client and server might agree to a 128-bit key that will be used to protect an RPC session using AES-GCM. The handshake consists of four serialized Protocol Buffer messages, an overview of which can be seen in Figure 6.

1. The client initiates the handshake by sending a `ClientInit` message. This message contains the client's handshake certificate, and a list of the handshake-related ciphers and record protocols the client supports. If the client is attempting to resume a terminated session, it will include a resumption identifier and encrypted server resumption ticket.

[Figure 6]

ALTS Handshake Protocol Messages



2. On receipt of the `ClientInit` message, the server verifies the client certificate. If valid, the server chooses a handshake cipher and record protocol from the list provided by the client. The server uses a combination of the information contained in the `ClientInit` message and its own local information to compute the DH exchange result. This result is used as an input to Key Derivation Functions[13] along with the transcript of the protocol to generate the following session secrets:

- A **record protocol** secret key *M* used to encrypt and authenticate payload messages,
- A **resumption secret** *R* to be used in a resumption ticket in future sessions,
- An **authenticator secret** *A*.

The server sends a `ServerInit` message containing its certificate, the chosen handshake cipher, record protocol, and an optional encrypted resumption ticket.

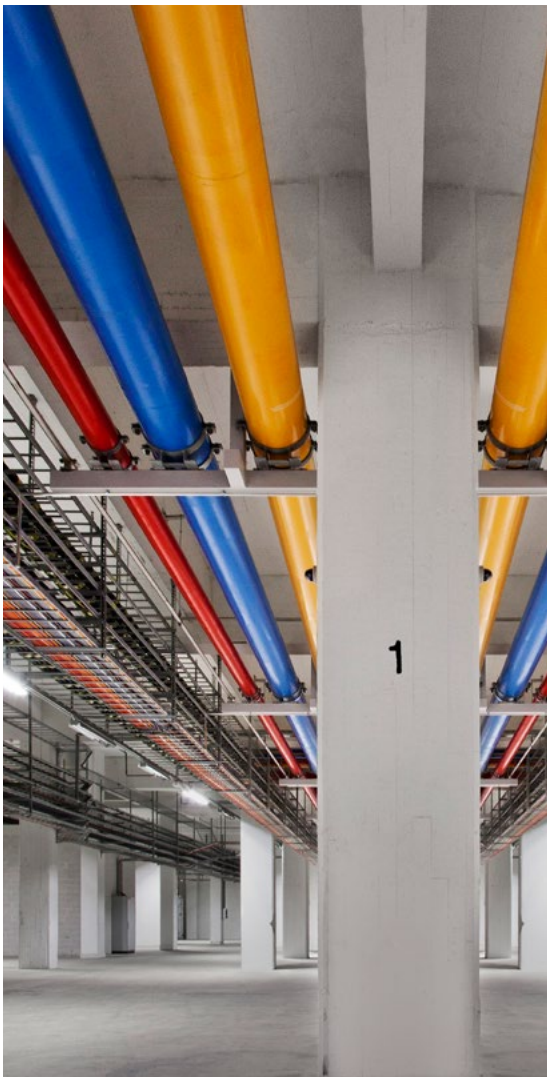
3. The server sends a `ServerFinished` message containing a handshake authenticator[14]. The value for this authenticator is calculated using a Hash-based Message Authentication Code (HMAC) computed over a pre-defined bit string and the authenticator secret *A*.

4. Once the client receives `ServerInit`, it verifies the server certificate, computes the DH exchange result similar to the server, and derives the same *M, R*, and *A* secrets. The client uses the derived *A* to verify the authenticator value in the received `ServerFinished` message. At this point in the handshake process, the client can start using *M* to encrypt messages. As the client is now capable of sending encrypted messages, we can say that ALTS has a one RTT handshake protocol.

[13] Specifically, HKDF-Extract and HKDF-Expand as defined in RFC-5869.
[14] ALTS handshaker protocol implementation concatenates ServerInit and ServerFinished messages into a single wire payload.

5. At the end of the handshake, the client sends a `ClientFinished` message with a similar authenticator value (see step 3) computed over different pre-defined bit string. If needed, the client can include an encrypted resumption ticket for future sessions. Once this message is received and verified by the server, the ALTS handshake protocol is concluded and the server can start using *M* to encrypt and authenticate further payload messages.

The Handshake protocol was reviewed by Thai Duong from Google's internal security analysis team and formally verified using the Proverif tool[15] by Bruno Blanchet with the assistance of Martin Abadi.



4.2 Record Protocol

Section 4.1 described how we use the Handshake protocol to negotiate a Record protocol secret. This protocol secret is used to encrypt and authenticate network traffic. The layer of the stack that performs these operations is called the ALTS Record Protocol (ALTSRP).

ALTSRP contains a suite of encryption schemes with varying key sizes and security features. During the handshake, the client sends its list of preferred schemes, sorted by preference. The server chooses the first protocol in the client list that matches the server's local configuration. This method of scheme selection allows both clients and servers to have different encryption preferences and allows us to phase in (or remove) encryption schemes.

4.2.1 Framing

Frames are the smallest data unit in ALTS. Depending on its size, each ALTSRP message can consist of one or more frames. Each frame contains the following fields:

- **Length:** a 32-bit unsigned value indicating the length of the frame, in bytes. This 4-byte length field is not included as part of the total frame length.
- **Type:** a 32-bit value specifying the frame type, e.g., data frame
- **Payload:** the actual authenticated and optionally encrypted data being sent.

[15] ProVerif: Cryptographic protocol verifier in the formal model.

The maximum length of a frame is 1MB plus 4 length bytes. For current RPC protocols, we further limit the frame length as shorter frames require less memory for buffering. Larger frames could also be exploited by a potential attacker during a Denial of Service (DoS) attack in an attempt to starve a server. As well as limiting the frame length, we also restrict the number of frames that can be encrypted using the same record protocol secret *M*. The limit varies depending on the encryption scheme that is used to encrypt and decrypt the frame payload. Once this limit is reached, the connection must be closed.

4.2.2 Payload

In ALTS each frame contains a payload that is integrity protected and optionally encrypted[16]. As of the publication of this paper, ALTS supports the following modes:

- AES-128-GCM, AES-128-VCM: AES-GCM and AES-VCM modes, respectively, with 128-bit keys. These modes protect the confidentiality and integrity of the payload using the GCM, and the VCM schemes[17], respectively.

- AES-128-GMAC, AES-128-VMAC: these modes support integrity-only protection using GMAC and VMAC, respectively, for tag computation. The payload is transferred in plaintext with a cryptographic tag that protects its integrity.

At Google, we use different modes of protection depending on the threat model and performance requirements. If the communicating entities are within the same physical boundary controlled by or on behalf of Google, integrity-only protection is used. These entities can still choose to upgrade to authenticated encryption based on the sensitivity of their data. If the communicating entities are in different physical boundaries controlled by or on behalf of Google, and so the communications pass over the Wide Area Network, we automatically upgrade the security of the connection to authenticated encryption, regardless of the chosen mode. Google applies different protections to data in transit when it is transmitted outside a physical boundary controlled by or on behalf of Google, since the same rigorous security measures cannot be applied.

[16] Payload encryption is negotiated as part of the Record protocol in the handshake.
[17] The 128-bit AES-GCM scheme is based on NIST 800-38D, and AES-VCM is discussed in details in AES-VCM, An AES-GCM Construction Using an Integer-Based Universal Hash Function.

Each frame is separately integrity protected and optionally encrypted. Both peers maintain both request and response counters, which synchronize during normal operation. If the server receives requests that are out of order, or repeated, cryptographic integrity verification fails, dropping the request. Similarly, the client drops a repeated or mis-ordered response. Furthermore, having both peers maintain the counters (as opposed to including their values in the frame header) saves additional bytes on the wire.

4.3 Session Resumption

ALTS allows its users to resume previous sessions without the need to perform heavy asymmetric cryptographic operations. Session resumption is a feature that is built into the ALTS Handshake protocol.

The ALTS handshake allows clients and servers to securely exchange (and cache) resumption tickets which can be used to resume future connections[18]. Each cached resumption ticket is indexed by a Resumption Identifier (ID_R) that is unique to all workloads running with the same identity and in the same datacenter cell. These tickets are encrypted using symmetric keys associated with their corresponding identifiers.

The ALTS handshake allows clients and servers to securely exchange resumption tickets which can be used to resume future connections.

ALTS supports two types of session resumption:

1. **Server side session resumption:** a client creates and encrypts a resumption ticket containing the server identity and the derived resumption secret *R*. The resumption ticket is sent to the server at the end of the handshake, in the `ClientFinished` message. In future sessions, the server can choose to resume the session by sending the ticket back to the client in its `ServerInit` message. On receipt of the ticket, the client can recover both the resumption secret *R* and the server's identity. The client can use this information to resume the session.

   The ID_R is always associated with a identity and not with specific connections. In ALTS, multiple clients can use the same identity in the same datacenter. This allows clients to

[18] Session resumption involves lightweight symmetric operations only if ephemeral parameters are not involved.

resume sessions with servers that they may not have communicated with before, e.g. if a load balancer sends the client to a different server running the same application.

2. Client side session resumption: at the end of a handshake the server sends an encrypted resumption ticket to the client in the `ServerFinished` message. This ticket includes the resumption secret *R* and the client's identity. The client can use this ticket to resume a connection with any server sharing the same ID_R.

When a session is resumed, the resumption secret *R* is used to derive new session secrets *M'*, *R'* and *A'*. *M'* is used to encrypt and authenticate payload messages, *A'* is used to authenticate `ServerFinished` and `ClientFinished` messages, and *R'* is encapsulated in a new resumption ticket. Note that the same resumption secret *R* is never used more than once.

5. Tradeoffs

5.1 Key Compromise Impersonation Attacks

By design, the ALTS handshake protocol is susceptible to Key Compromise Impersonation (KCI) attacks. If an adversary compromises the DH private key, or the resumption key, of a workload they can use the key to impersonate other workloads to this workload[19].

[19] Key Agreement Protocols and Their Security Analysis.

This is explicitly in our resumption threat model, as we want resumption tickets issued by one instance of an identity to be usable by other instances of that identity.

There is a variant of the ALTS handshake protocol that protects against KCI attacks, but it would only be worth using in environments where resumption is not desired.

5.2 Privacy for Handshake Messages

ALTS is not designed to disguise which internal identities are communicating, so it does not encrypt any handshake messages to hide the identities of the peers.

5.3 Perfect Forward Secrecy

Perfect Forward Secrecy (PFS) is supported, but not enabled by default, in ALTS. We instead use frequent certificate rotation to establish forward secrecy for most applications. With TLS 1.2 (and its prior versions), session resumption is not protected with PFS. When PFS is enabled with ALTS, PFS is also enabled for resumed sessions.

5.4 Zero-Roundtrip Resumption

TLS 1.3 provides session resumption that requires zero roundtrips (0-RTT), however this has weaker security properties[20]. We decided not to include a 0-RTT option in ALTS because RPC connections at Google are generally long-lived. Consequently, reducing the channel setup latency was not a good tradeoff for the additional complexity and/or reduced security that 0-RTT handshakes require.

6. Further References

For information on how Google encrypts data in transit, see our Encryption in Transit in Google Cloud whitepaper.

For an overview of how security is designed into Google's technical infrastructure, see our Google Infrastructure Security Design Overview.

[20] Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates.